STUDENT HANDBOOK FOR STANDARD X

ARTIFICIAL INTELLIGENCE, DATA ANALYTICS AND MACHINE LEARNING

मध्य प्रदेश राज्य मुक्त स्कूल शिक्षा बोर्ड, भोपाल

Table of Contents

Introduction 8

Student Learning Approach 9

Module 1: Artificial Intelligence and its Utility 11

Chapter 1: AI and its Applications 12

1.1 Natural Language Generation (NLG) 12

1.2 Exposure to AI and its Applications Across Industry 14

1.3 Conversational AI 16

1.4 Machine Learning and Intelligent Edge 17

1.5 Assess Your Knowledge 17

1.5.1 Questions for Assessment 17

1.5.2 Fill in the Blanks 17

1.5.3 Questions to Ponder upon 18

1.5.4 Links for Further Reading 18

1.6 Reference Links 18

Module 2: Building Foundations of Python 19

2.1 Python - Basic Syntax 20

2.2 Python - Variable Types 20

2.3 Assigning Values to Variables 21

2.4 Python - Basic Operators 21

2.5 Assess Your Knowledge 23

2.5.1 Practical Assignment and Lab Work 23

2.5.2 Questions to Ponder Upon 24

2.5.3 Links for Further Reading 24

2.6 Reference Links 24

Module 3: Overview of Azure 25

Chapter 3: Overview of Azure 26

3. 1 Introduction to the Lab 26

3.2 Set Up your Azure Subscription 27

3.3 Install Project Template in Visual Studio 28

3.4 Download the Lab materials 29

3.5 Install Bot Framework Emulator 29
3.6 Create a BOT Resource in Azure 30
3.7 Assess Your Knowledge 31
3.7.1 Practical Assignment and Lab Work 31
3.8 Reference Links 31
Module 4: Quickly Create a Basic BOT 32
Chapter 4: Overview of Basic BOT 33
4.1 Create a Project 33
4.2 Debugging with Bot Framework Emulator 33
4.3 Update Welcome Message 34
4.4 Assess Your Knowledge 35
4.4.1 Practical Assignment and Lab Work 35
4.5 Reference Links 35
Module 5: Adding LUIS to your BOT 36
Chapter 5: Overview of LUIS 37
5.1 Introduction to LUIS 37
5.2 Create a LUIS Subscription 37
5.3 Import and Extend the LUIS Model 38
5.4 Install LUIS Package 40
5.5 Add the LUIS Recognizer to your BOT 41
5.6 Adjust your BOT to process LUIS results 41
5.7 Test LUIS Configuration 43
5.8 Assess Your Questions 43
5.8.1 Practical Assignment and Lab Work 43
5.9 Reference Links 43
Module 6: Adding Advanced Conversational Features to your BOT 44
Chapter 6: Overview of BOT Framework Emulator 45
6.1 Add a Visual Response to your BOT: Carousel 45
6.2 Test the Visual Response 46
6.3 Install and Set Up Dialogs Package 47
6.4 Set Up the Conversation Flow 49
6.5 Test the Conversation Flow 58

6.5 Assess Your Knowledge 59
6.5.1 Practical Assignment and Lab Work 59
6.6 Reference Links 59
Module 7: Adding Knowledge to your BOT Using QnA Maker 60
Chapter 7: Overview of QnA Maker 61
7.1 Introduction to QnA Maker 61
7.2 Set Up QnA Maker 61
7.3 Set Up your QnA Maker Instance 62
7.4 Install QnA Maker Package 63
7.5 Assess Your Knowledge 63
7.5.1 Practical Assignment and Lab Work 63
7.6 Reference Links 63
Chapter 8: Integration of QnA Maker in BOT 64
8.1 Add QnA Maker to the BOT 64
8.2 Test QnA Maker in Bot Emulator 66
8.3 Assess Your Knowledge 66
8.3.1 Practical Assignment and Lab Work 66
8.4 Reference Links 66
Module 8: Implementing Personality Chat 67
Chapter 9: Overview of Personality Chat 68
9.1 Overview 68
9.2 Add Project Personality Chat to the BOT 68
9.3 Test Personality Chat in Bot Emulator 69
9.4 Assess Your Knowledge 70
9.4.1 Practical Assignment and Lab Work 70
9.5 Reference Links 70
Chapter 10: Adding Speech Support 71
10.1 Overview 71
10.2 Create a Speech Subscription 71
10.3 Set Up Web Chat 72
10.4 Add Text to Speech to BOT 73
10.5 Deploy to Azure from Visual Studio 75

10.6 Test in Web Chat 76
Chapter 11: Overview of Custom Speech Model 77
11.1 Overview 77
11.2 Speech Customisation: Using Custom Speech 77
11.3 Customise Language Model and Pronunciation 78
11.4 Create Custom Speech-to-Text Endpoint 79
11.5 Modify Chatbot to Support Custom Speech 80
11.6 Test Custom Speech in Web Chat 80
Chapter 12: Adding Speech Translation to your BOT 82
12.1 Overview 82
12.2 Create a Translator Text Subscription 83
12.3 Add Translation Support to the BOT 83
12.4 Test the Speech Translation 85
Chapter 13: Adding a BOT to Web Application 86
13.1 Add the Web Chat Libraries 86
13.2 Add the Web Chat Widget 86
13.3 Add the Initialisation Script 86
13.4 Test the BOT Using the Web Chat Widget 88
Chapter 14: Consuming a Custom Voice 89
14.1 Create a Voice Font 89
14.2 Add the Voice Font to your BOT 90
Module 9: Create LUIS App without Coding 91
Chapter 15: Web-based LUIS 92
15.1 Introduction to LUIS 92
15.2 Define your App 93
15.3 Train and Test the Model 95
15.4 Publish the App and Generate JSON Code 97
15.5 Assess Your Knowledge 99
15.5.1 Practical Assignment and Lab Work 99
15.6 Reference Links 99
Module 10: Project Work 100
Practical Assignments/ Project Work 101

Glossary 103

Disclaimer:

Copyright © 2019 by Microsoft Corporation India Private Limited.

All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of Microsoft Corporation India Private Limited.

Introduction

Artificial intelligence (AI) is a vast subject, which studies human behaviour and the various ways it can be mapped on to a computer to demonstrate human-like behaviour. Eventually the objective of AI is to create machines that can demonstrate human-like behaviours, such as learn, plan and solve problems as humans do with minimal human interference. Though, we are yet to build a machine that is at par with a human in its complexity of cognitive abilities, we have come a long way from the time the concept of AI was formed. In the past, we have created certain AI-based applications that have a faster data processing speed and have proven to be more intelligent than humans. The Deep Blue system, which defeated the reigning grandmaster at Chess, is a well-documented example.

In this handbook, our focus is to understand the basics of AI and its applications in the real world. We will introduce some basic algorithms aiding comprehension and learning on problem definition, data handling, representation of knowledge and recognition of patterns. We will also understand cloud technology and its working in an in-depth manner to facilitate comprehension.

The main goal of this course is to offer not only the basics of AI, but also facilitate learning in the student with the help of real-life examples, practicals to expedite kinaesthetic learning and real-life simulation scenarios. These modes of learning have been used keeping in mind that the student is able to comprehend the thought behind AI and the creation of related applications in its entirety. We hope students who develop further interest in the subject, and wish to pursue it further, may enrol for advanced study at a graduate level.

Student Learning Approach

Learning takes place in an all-encompassing holistic manner and, thus, needs to integrate both conceptual and practical approaches to learning. Children tend to learn in a not-so linear manner, wherein, revisiting concept helps them to not only understand better, but also reinforces concepts and connections between them.

Learning is a process which establishes connections, joining the dots to create a link between the observed facts and the experiments done, thereafter, to reinforce the realisation gained. Learning is an on-going process, which is not restricted to an educational institution; rather, its essence is carried even at home and other facets of life to aid the process. Hence, learning cannot occur in a linear progression. Assessments are complex in nature, as they test a variety of functions and are aimed at catering to the overall test of both theoretical knowledge as well as practical applications.

There are different modes of assessment that particularly highlight the investigative approach to learning, application of the learning to develop skills and knowledge application to solve a multitude of real-life problems. It is also designed to create a thorough understanding of the topic mentioned and beyond to aid comprehension, analysis, application, evaluation of problem at hand and devising a suitable solution for the same. This is in congruence with the higher order skills of a reworked Bloom's taxonomy.

The curriculum is designed keeping in mind the updates on various rules and regulation laid by the different education boards and the Ministry of Education. The entire syllabus is designed to aid three levels of assessment namely individual assessment, peer/group assessment and self-assessment for a complete evolution of the child. Teaching–Learning–Assessment is a complete process that every student has to undergo. In order to complete the process, it is necessary that all aspects of learning are offered to the students in order to achieve the goal.

At the end of this curriculum, you will be able to:

- Understand the role of BOT-to-people interactions.
- Understand how a BOT can accomplish many tasks in more natural ways.
- Understand how speech-enabled BOTs can overcome speech recognition barriers such as speaking style, vocabulary and background noise, and can also responding to the user in a natural, familiar voice.
- Know the fundamentals of AI programming techniques and advanced machine learning in a modern programming language.
- Easily start building a sophisticated BOT able to perform several intelligent actions.

Module 1: Artificial Intelligence and its Utility

Chapter 1: AI and its Applications

The students are required to understand how AI has been instrumental in the simplification of tasks that consume human energy and time, and are error-prone at times. The various sub-topics have been introduced with the aim to develop and arouse interest in the young minds to explore new possibilities that they can have with AI. The students are to feel encouraged to explore how AI enables machine learning and solves day-to-day problems.

At the conclusion of the module, the students should be able to:

- Understand the concept of artificial intelligence and its various applications in different fields.
- Align the concepts to real-life situations to comprehend the applications.
- Create simple, yet effective, solutions for driving business value from intelligent predictions, which require connecting AI systems to business processes.

Artificial Intelligence or AI tends to have some impressive capabilities as we have seen in self-driving vehicles or humanoids, as many robots with AI capabilities are known as. In recent times, the machines have advanced powers of computation, thus, taking in to consideration all the probabilities of a situation happening or not. These machines also possess almost human-like intelligence to learn and recognise patterns in data to facilitate learning.

Though we do not have a fully functional AI-based robot with us on a full-time working however, it seems just a matter of time that we have a fully-functional humanoid capable of working with us or even staying with us at our homes. Here are a few applications, which we can regularly use for the machine to be as important and engaging in humans can be in real-life.

1.1 Natural Language Generation (NLG)

It is a sub-discipline of AI that recognises human language in text form in order to convert it into a form, which will enable the computer system to communicate with efficiency and accuracy. It is used for report generation and creation of market summaries.

The machine is designed to analyse human speech with all the nuances and can analyse, comprehend and generate an automated version of linguistic activities and communication methods. For example, the NLP program can extract relevant texts from the news and other official announcements to create a pattern into an algorithmic trading decision software and create learning from the data.

Another important use of NLP that we can observe is that of classification of the text. This is specifically used in spam control of emails. There are several email providers who use similar NLP-based services to look at the content of the emails and other technicalities associated with a mail to understand whether the mail is genuine or from a fake person. NLG can help you to do the following:

- Manually look up data, compare or scan through tables.
- Analyse charts, metrics or maps.
- Compute, interpret and communicate performance.
- Develop recurring status reports.
- Open multiple screens or refer to different applications.

Speech Recognition

This is a technology, wherein, the systems capture the natural conversation by human(s) and convert the same into instruction to perform the given tasks. These are used in voice-response interactive systems and mobile apps. For example, android devices and iPhones have built-in applications to convert speech into text. The technology of recognising speech is also used in the medical world, defence forces and even in robotics (Lifewire, 2019).

However, there are times when speech recognition does not work the way it should. This happens at times when different people pronounce the same word in a different manner altogether. Another scenario, wherein, the software may fail is when there is great background noise. The software will pick up the noise and may not be able to interpret it well.

Virtual Agents

A virtual agent is a computer program that is designed to interact with humans. You can see many of them in the chatbots that you interact with, as well as smart home management apps.

Gone are the days when the customer would realise chatting with a BOT. In the current scenario, with the help of both artificial intelligence and machine learning taken together, the BOT is designed to come up with a specific and customised response to the queries of a customer. From maintaining a contact centre to catering to various business functions such as finance and accounting, HR and IT helpdesk, virtual agentsno longer perform basic functions but also take care of complex situation-based query solving (IPsoft, 2019).

1.2 Exposure to AI and its Applications Across Industry

Entertainment

Facebook messenger chatbot named MojiHunt is followed by more than 20,000 users. The chatbot helps users to play games using stickers, especially, when they want to kill time or are in line to wait for a service or a product. Another popular chatbot is TriviaBlast, which asks questions to the users. The result and scores can be shared online and you can even compete with your friends.

News

Chatbots are quite smart as they understand the preference of a user and send similar news on related topics to the user on their social media feed or other media handles. The chatbot from CNN is quite famous for delivering quality output in terms of relevant news to the users. One more cool and popular news chatbot is theScore. This BOT is for people who are interested in sports and want to know the latest sports events. Also, you can follow your favourite teams and leagues.

Health HealthTap is a chatbot that will simply tell you sure signs and symptoms of a particular illness and offer your doctor's recommendation. Florence is another chatbot, which will check your symptoms, send you daily health tips, recommend a doctor and tell you a lot more about an illness. You have to note that Florence will only recommend a doctor, however, it doesn't offer you medical recommendation.

Deep Learning Platforms

Deep learning platforms use neural networks with layers of abstraction to copy the working of the human brain. It is designed to process data and create patterns for making decisions. It is currently being used to identify patterns and categorise applications that are only compatible with large-scale data sets.

Biometrics

This technology can identify, measure and analyse human behaviour and physical aspects of the body's structure and form. It allows for more natural interactions between humans and machines, including interactions related to touch, image, speech and body language recognition, and is big within the market research field. It is often used as recognition software for many high-end applications both for public consumption and industrial applications.

Robotic Processes Automation

Robotic processes automation makes use of scripts and methods that copy and systematise human tasks to support large scale processes. It is, particularly, useful for situations when hiring humans for a specific job or task is too expensive or inefficient. It is used in areas that save businesses from dedicating hours to mechanical and iterative tasks.

1.3 Conversational AI

Conversational Artificial Intelligence or Conversational AI is a set of technologies that enable computers to simulate conversations. This form of AI is preferred because it is time-bound and quite engaging. The element of conversation keeps the customer engaged and, in the background, it can simultaneously work on the customer delight aspect of the transaction. The amount of queries that can be handled by a business can be quite a lot as the query resolution time is lesser and queries can be solved effectively.

Natural Language Processing (NLP) is an application of artificial intelligence. It enables the machines to both process and comprehend human language in the way it is written. In the recent times, deep learning models can not only analyse large volumes of text but can also come up with services like summarisation of text, language translation, context modelling and even sentiment analysis (Mady Mantha, 2019). Natural Language Understanding (NLU) is a branch of NLP that converts the natural language spoken by humans into structured data. It can perform two tasks- intent classification and entity extraction.

An intent of a user is what a user is trying to convey or accomplish. For example, when we read a sentence, we immediately understand the meaning or intent behind that sentence. Intent classification is a two-step process. First, we feed an NLU model with labelled data that provides the list of known intents and example sentences that correspond to those intents. Once trained, the model is able to classify a new sentence that it sees into one of the predefined intents.

Entity extraction is the process of recognising key pieces of information in a given text. Things like time, place and name of a user all provide additional context and information related to an intent. Intent classification and entity extraction are the primary drivers of conversational AI.

1.4 Machine Learning and Intelligent Edge

Intelligent Edge may be defined as a set of connected systems and devices that gather and analyse information — getting ready to the registered users, the data or both. Users get time period insights and experiences, delivered by extremely responsive and contextually aware apps. Unarguably, the Intelligent Edge provides analytics capabilities that were erstwhile confined to on-premises or cloud information centres (Adnan Khan, 2019).

The intelligent cloud is a model of seamless computing, which is enabled by the public cloud and artificial intelligence (AI) technology combined together, for a variety of intelligent applications and systems that can be thought of. The term 'Intelligent Edge' or 'Intelligence at the Edge' is used in many ways, by the industries. It can be understood as a producing floor, building, campus, city, house, crop field, powerhouse, an influence plant, oil rig, field house, piece of land, in your automotive, in the sky or beneath the ocean. It's all over and it's wherever the 'things' square measure within the web of Things (IoT).

1.5 Assess Your Knowledge

1.5.1 Questions for Assessment

1.5.2 Fill in the Blanks:

1. A Virtual Agent is a computer program that is designed to interact with humans.
2. Biometrics is a technology that is designed to identify or measure human body parameters to be used for access control.
3. Facebook messenger chatbot named MojiHunt helps the users to play games.
4. Florence is a chatbot.

- Give some examples of Conversational AI.
- Where can we use conversational AI? Quote an example. What do you mean by natural language detection?

1.5.3 Questions to Ponder upon

- Suggest various practical applications of NLP.
- How can AI enhance biometrics and other security measures?
- Can AI fool biometric scanners?

1.5.4 Links for Further Reading

- https://hub.packtpub.com/intelligent-edge-analytics-7-ways-machine-learning-is-driving-edge-computing-adoption-in-2018/
- https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-conversation.html
- https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-conversation.html

1.6 Reference Links

- Adnan Khan. 2019. Machine Learning Model Optimization for Intelligent Edge. [ONLINE] Available at: https://medium.com/swlh/machine-learning-model-optimization-for-intelligent-edge-d0f400111002. [Accessed 05 July 2019].
- IPsoft. 2019. What is a Virtual Agent? . [ONLINE] Available at: https://www.ipsoft.com/2019/02/01/what-is-a-virtual-agent/. [Accessed 05 July 2019].
- Lifewire. 2019. What Is Speech Recognition?. [ONLINE] Available at: https://www.lifewire.com/what-is-speech-recognition-3426721. [Accessed 05 July 2019].
- Mady Mantha. 2019. Conversational AI: Design & Build a Contextual AI Assistant. [ONLINE] Available at: https://towardsdatascience.com/conversational-ai-design-build-a-contextual-ai-assistant-61c73780d10. [Accessed 05 July 2019].

Module 2: Building Foundations of Python

Chapter 2: Understanding Python

Students are required to understand the basic fundamentals of Python. Students who develop interest in the subject or want to learn more about advanced possibilities can explore further. At the conclusion of the module, the students should be able to:

- Understand the basic concepts of Python and be able to write basic programs in the language.
- Well-versed with the numerous functionalities of the language and the implementations of the various modules to create the code to build a BOT.
- Able to create simple solutions using Python for age-appropriate level.

2.1 Python - Basic Syntax

Python is a high-level programming language, which is interpreted (runs via an interpreter) for a purpose. The language is interactive and object-oriented. It was created by Guido van Rossum and like Perl, the source code of the language, is available under the GNU General Public License (GPL).

Python files have extension .py. Type the following source code in a test.py file:

```
print "Hello, Python!"
```

```
$ python test.py
```

This produces the following result:

```
Hello, Python!
```

2.2 Python - Variable Types

Standard Data Types

There are five types of standard data types while data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the

operations possible on them and the storage method for each of them.

Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

The following link gives a complete insight on using the standard data types of Python, students to go to the link and understand and practice the same:
https://www.learnpython.org/en/Conditions

2.3 Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the '=' operator is the name of the variable and the operand to the right of the '=' operator is the value stored in the variable. For example:

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

2.4 Python - Basic Operators

Operators are the constructs, which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and '+' is called operator.

Types of Operator- Python language supports the following types of operators:

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

The following link gives a complete insight on using the basic operators of Python, students to go to the link and understand and practice the same:
https://www.tutorialspoint.com/python/python_basic_operators.htm#

Python- Decision Making

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE.

Following is the general form of a typical decision making structure found in most of the programming languages:

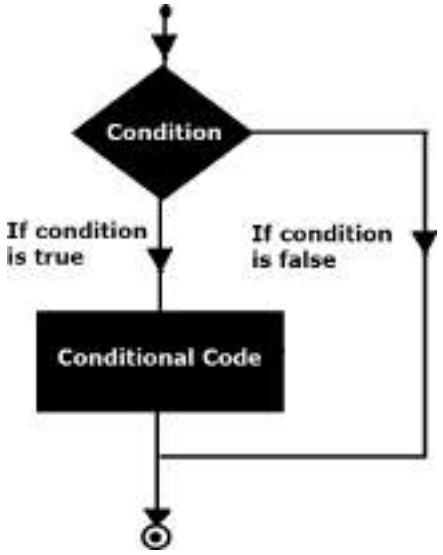


Fig 2.1 Decision Making Structure

Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

The following link gives a complete insight on decision making in Python, students to go to the link and understand and practice the same:
https://www.tutorialpoint.com/python/python_decision_making.htm

2.5 Assess Your Knowledge

2.5.1 Practical Assignment and Lab Work

- What is the result of 10 == “10”?
- What is the result of “bag” > “apple”?
- What is the result of not (True or False)?
- Under what circumstances does the expression 18 <= age < 65 evaluate to True?
- What does range (1, 10, 2) return?
- Name three iterable objects in Python.

2.5.2 Questions to Ponder Upon

1. How is Python interpreted?
2. How is memory managed in Python?
3. What are the tools that help to find bugs or perform static analysis?
4. What are Python decorators?
5. What is the difference between a list and tuple?

2.5.3 Links for Further Reading

- https://hub.packtpub.com/intelligent-edge-analytics-7-ways-machine-learning-is-driving-edge-computing-adoption-in-2018/
- https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-conversation.html
- https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-conversation.html

2.6 Reference Links

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.python.org/

Module 3: Overview of Azure

Chapter 3: Overview of Azure

- Through this chapter we aim to make the students:Familiar to the Azure platform.
- Understand the working of the Lab and its subsequent features.
- Assemble and incorporate components to create a chatbot.

3. 1 Introduction to the Lab

AI plays an ever-increasing role in BOT-to-people interactions. Whether you want to make a dinner reservation at a restaurant or simply ask for information about a store over the phone, a BOT can help you to accomplish many tasks in natural ways. Sophisticated conversational AI agents now allow users to interact with a wealth of services such as natural speech language or enhanced chat styles. For instance, speech-enabled BOTs can overcome speech recognition barriers such as speaking style, vocabulary and background noise, and can also respond to the user in a natural, familiar voice as well. Intelligent agents can perform contextual conversations, with even personality layers based on the type of scenario identified: this can minimise fallback responses such as "I don't understand", but also grounds small talk in a personality and consistent tone.

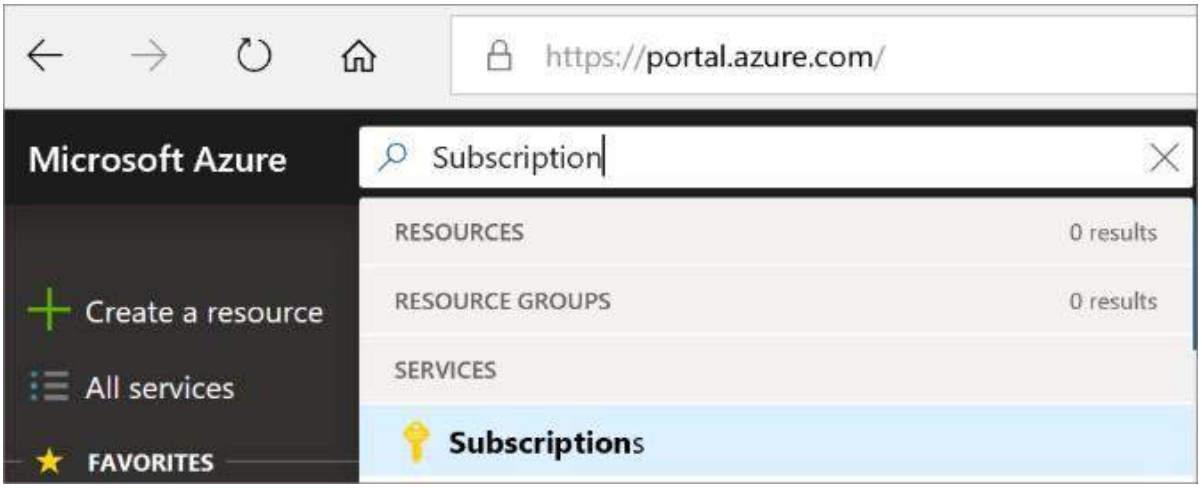
With the Conversational AI technologies in these various steps, you'll be able to easily start building a sophisticated BOT, which can perform several intelligent actions: from capturing speech (even in difficult, noisy environments), translate it if necessary, process it into intents, reply back not just with text-based chat but also with voice-enabled answers. Another thing to bear in mind is that it is important to ensure you build it responsibly. This includes a number of different aspects ranging from transparency, diversity, reliability, security and more.
Microsoft has created guidelines for responsibly building conversational AI. See our Responsible Conversational AI learning path at https://aischool.microsoft.com/responsible-ai/learning-paths/responsible-conversational-ai/responsible-conversational-ai to find out more.

3.2 Set Up your Azure Subscription

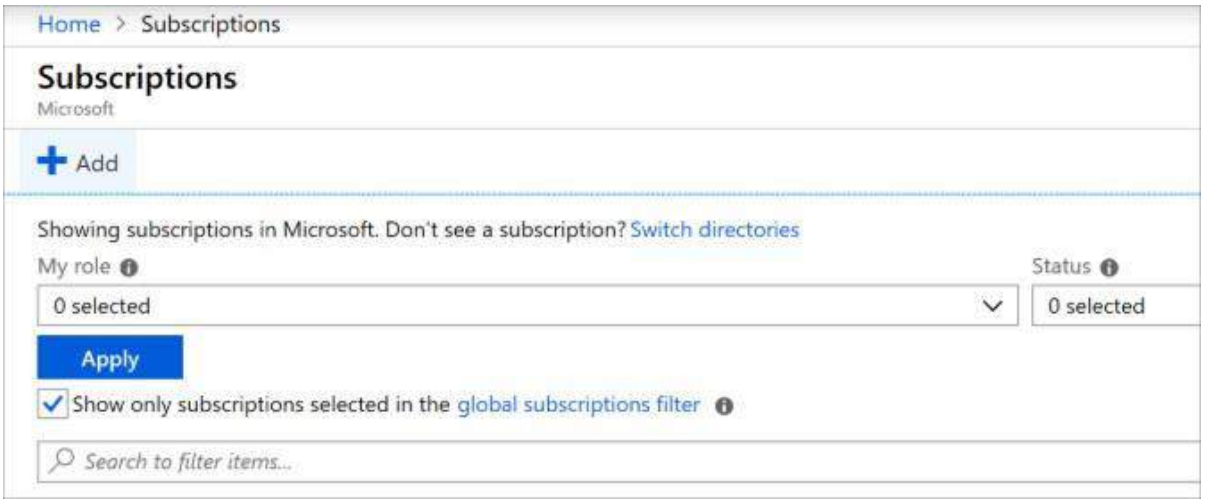
Here is a series of steps that can be followed to set up your Azure platform Login, successfully:

Step 1: Sign in to the Azure Portal at https://portal.azure.com/

Step 2: Search for Subscription.



Step 3: Select Add.



Step 4: If you have access to multiple billing accounts, select the billing account for which you want to create the subscription.

Step 5: Fill the form and click Create. The tables below list the fields on the form for each type of billing account.

Following are the fields and their definitions to understand the kind of information that each of these field captures.

Enterprise Agreement

Field	Definition
Name	The display name that helps you easily identify the subscription in the Azure portal.
Offer	Select EA Dev/Test, if you plan to use this subscription for development or testing workloads else use Microsoft Azure Enterprise. DevTest offer must be enabled for your enrollment account to create EA Dev/Test subscriptions.

Microsoft Customer Agreement

Field	Definition
Billing profile	The charges for your subscription will be billed to the billing profile that you select. If you have access to only one billing profile, the selection will be greyed out.
Invoice section	The charges for your subscription will appear on this section of the billing profile's invoice. If you have access to only one invoice section, the selection will be greyed out.
Plan	Select Microsoft Azure Plan for DevTest, if you plan to use this subscription for development or testing workloads else use Microsoft Azure Plan. If only one plan is enabled for the billing profile, the selection will be greyed out.
Name	The display name that helps you easily identify the subscription in the Azure portal.

Microsoft Partner Agreement

Field	Definition
Customer	The subscription is created for the customer that you select. If you have only one customer, the selection will be greyed out.
Reseller	The reseller that will provide services to the customer. This is an optional field, which is only applicable to indirect providers in the CSP two-tier model.
Name	The display name that helps you easily identify the subscription in the Azure portal.

3.3 Install Project Template in Visual Studio

1. We will use Visual Studio 2017 with BOT template to develop a BOT. If you don't have Visual Studio you can download it from the following URL given below:

    - Download Visual Studio 2017 15.6.7 or newer (any edition) from https://www.visualstudio.com/downloads/.

- Refer Visual Studio 2017 system requirement from https://www.visualstudio.com/en-us/productinfo/vs2017-system-requirements-vs.

2. Open Visual Studio and click on Tools from the menu.
3. Click on Get Tools and Features. Make sure your Visual Studio has the following workloads:

    - ASP.NET and web development.
    - Azure development.
    - .NET Core cross-platform development.

    Note: Install any of the missing workloads as they are required by the Bot Builder SDK v4.
4. Open Microsoft Edge and navigate to https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4.
5. Click Download.
6. When prompted, click Open.
7. Click Modify and complete the template installation.

3.4 Download the Lab materials

1. Open Microsoft Edge and navigate to GitHub.
2. Click Clone or download.
3. You can clone the repo using git or click Download ZIP to directly download the code from your browser.

[!ALERT] Make sure to uncompress/clone your code into Downloads/ailabs.

1. Copy the contents of the BuildAnIntelligentBot folder to Downloads/buildanintelligentbot.

3.5 Install Bot Framework Emulator

1. Open Microsoft Edge and navigate to https://github.com/Microsoft/BotFramework-Emulator/releases.
2. Download the exe file of the latest *4.x* version available.
3. When prompted, click Open.
4. Complete the installation.

3.6 Create a BOT Resource in Azure

1. Log into the Azure Portal(portal.azure.com).
2. Click Create Resource [+] from the left menu and search for Web App Bot.
3. Select the first result and then click the Create button.
4. Provide the required information:

    - Bot name: chatbot-<your initials>
    - Create a new resource group with the name: ttmb-lab-<your initials>.
    - Location: West US
    - Pricing tier: F0 (10K Premium Messages)
    - App name: chatbot-<your initials>
    - Bot template: Basic C#

        Note: This BOT template includes Language Understanding and it will create a LUIS application that we will use later in the lab.

    - LUIS App location: West US
    - Azure Storage: Create a new one with the recommended name.
    - Application Insights Location: West US 2
5. Click on App service plan/Location.
6. Click Create New.
7. Provide the required information:

    - App Service plan name: chatbot-<your initials>
    - Location: West US
8. Click OK to save the new App service plan.
9. Click Create to deploy the service. This step might take a few moments.

10. Once the deployment is completed, you will see a Deployment succeeded notification.
11. Go to All Resources in the left pane and search for the new resource (chatbot-<your initials>).
12. Click on the Web App Bot to open it.
13. Click on the Test in Web Chat option in the menu to test the new BOT.
14. Type Hello into the built-in chat control, you should get a response from your BOT.

3.7 Assess Your Knowledge

3.7.1 Practical Assignment and Lab Work

- In groups of three or more, as decided by the faculty, create a BOT resource in Azure to accomplish a simple task of your choice.

3.8 Reference Links

- https://www.visualstudio.com
- https://aischool.microsoft.com

Module 4: Quickly Create a Basic BOT

Chapter 4: Overview of Basic BOT

Through this chapter we aim to make the students:

- Successfully create a basic structural BOT using minimum instructions and the various built-in features.

4.1 Create a Project

1. Open Visual Studio 2017 from the Start Menu.
2. Click Create new project....You can also find this option in VS Menu: File > New > Project.
3. In the search box from the top right corner type: bot.
4. Select Echo Bot (Bot Framework v4).
5. Provide a name for your new project: MyBot.
6. Click Ok.

> Alert: Do not name the project EchoBot, as there will be conflicts with the EchoBot class in Bots/EchoBot.cs. Note: The template has the latest version of the new Bot Framework SDK pre-installed.

4.2 Debugging with Bot Framework Emulator

1. Open the Bots/EchoBot.cs file.
2. Put a breakpoint on line 18.
3. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).

> Note: A new web page with the Web Chat will be opened in your browser. Don't use this one yet, as we'll configure it later.

4. Once the page has loaded in Microsoft Edge, copy the site host from the address bar to the clipboard (e.g. http://localhost:XXXX).

> Note: We won't be using the web site chat control until later in the lab. Make sure you use the emulator until then!

5. Open the Bot Framework Emulator from the Start Menu.

6. Click File -> New bot configuration.
7. Paste the URL that you copied from your browser, and add /api/messages at the end of it. It should look something like: http://localhost:3978/api/messages.
8. Click the Save and Connect button to connect to the local BOT and start a conversation.

   Note: You will need to save the configuration to a file before; just select a location, enter the name and save.
9. Type Hello and press enter.
10. Return to Visual Studio and wait for the breakpoint to be hit.
11. Mouse over the context.Activity.Text variable to see your input message.
12. Press Continue in the toolbar.
13. Remove the breakpoint.
14. Go back to the emulator and see the response from the BOT.
15. Stop debugging by clicking the stop button in Visual Studio's toolbar.

4.3 Update Welcome Message

1. Open the Bots/EchoBot.cs file and add the following import at the beginning of the file:

2. `using System.Linq;`
3. Modify the OnMembersAddedAsync method, replace the text Hello and Welcome! with the following text:

4. `"Hi! I'm a restaurant assistant bot. I can add help you with your reservation."`

5. Let's run the BOT to see the welcome message.

   - Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
   - Return to the Bot Framework Emulator.
   - Click the Restart conversation button to start a new conversation.

- See the updated welcome message displayed at the beginning of the conversation.
- Stop debugging by clicking the stop button in Visual Studio's toolbar.

4.4 Assess Your Knowledge

4.4.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, create a BOT resource in Azure to accomplish a simple task of your choice

4.5 Reference Links

- https://www.visualstudio.com
- https://aischool.microsoft.com

Module 5: Adding LUIS to your BOT

Chapter 5: Overview of LUIS

Through this chapter we aim to make the students:

- Understand the working of the Language Understanding Service.
- Successfully incorporate the features in the BOT.Test the incorporation and subsequent deployment of the features.

5.1 Introduction to LUIS

Language Understanding (LUIS) is an API service based on the cloud system, which applies custom machine learning intelligence to a user's conversational, natural language text. This is to predict the overall meaning in the given context and pull out relevant, detailed information. A client application for LUIS is any conversational application that communicates with a user in natural language to complete a task. Examples of client applications include social media apps, chatbots and speech-enabled desktop applications.

5.2 Create a LUIS Subscription

Create the LUIS resource in Azure:

1. Return to the Azure Portal (portal.azure.com).
2. Click Create Resource [+] from the left menu and search for Language Understanding.
3. Select the first result and then click the Create button.
4. Provide the required information:
    - App name: chatbot-luis-<your_initials>.
    - Location: West US.
    - Pricing tier: F0 (5 Calls per second, 10K Calls per month).
    - Use existing resource group: ttmb-lab-<your initials>.
5. Click Create. This step might take a few seconds.
6. Once the deployment is complete, you will see a Deployment succeeded notification.
7. Go to All Resources in the left pane and search for the new resource (chatbot-luis-<your initials>).
8. Click on the resource.

9. Go to the Keys page.
10. Copy the Key 1 value into Notepad.

Note: We'll need this key later on.

5.3 Import and Extend the LUIS Model

1. Login to the LUIS portal.

Note: Use the same credentials as you used for logging into Azure.

2. If this is your first login in this portal, you will receive a welcome message. Follow the next steps to access the LUIS dashboard:
    - Scroll down to the bottom of the welcome page.
    - Click Create LUIS app.
    - Select United States from the country list.
    - Check the I agree checkbox.
    - Click the Continue button.
3. From My Apps, look for the following app chatbot-<your initials>. You will see an existing LUIS App (chatbot--), this was generated by the Web App Bot deployment.
4. Click on the Manage option.
5. Copy the LUIS Application ID to Notepad.

Note: We'll need this app ID later on.

6. Click the Versions option.
7. Click on Import version.
8. Click on Choose version file.
9. Select the base model from Downloads\buildanintelligentbot\resources\talk-to-my-bot.json.
10. Click on the Done button.
11. Wait for the import to complete.
12. Click on the Train button and wait for it to finish.

13. Click the Test button to open the test panel.
14. Type I need a dinner reservation and press enter.

    Note: It should return the ReserveTable intent.

15. Click the Test button in the top right to close the test panel.
16. Add a new intent:

    - Click on Build.
    - Click on Intents.
    - Click on Create new intent.
    - Type the new intent name: TodaysSpecialty
    - Click Done.

17. Add a new utterance by typing the following example in the textbox:

    - Type: what is the specialty for today?
    - Press Enter.

18. Add another new utterance:

    - Type: what's the dish of the day?
    - Press Enter.

19. Test your new intent:

    - Click on the Train button and wait for it to finish.
    - Click on Test button.
    - Type the following test utterance: what's today's special?
    - Press Enter.

    Note: The test should return the TodaysSpecialty intent.

20. Click on the Manage option.
21. Click the Keys and Endpoints option.
22. Copy the Authoring Key to Notepad.

    Note: We'll need this key later on.

23. Click on + Assign resource. You might need to scroll down to find the option.

    - Select the only tenant.
    - Select your subscription.
    - Select the key of your LUIS resource.
    - Click on Assign resource.

24. Publish your application:

    - Click the Publish button.
    - Click on the Publish button next to the *Production* slot.
    - Wait for the process to finish.

5.4 Install LUIS Package

Before we continue adding services to our BOT, we'll open the sample code provided in order to have access to prebuilt services and models to speed up the lab.

1. Click File from Visual Studio menu.
2. Click Open Project/Solution.
3. Select the solution file Downloads\buildanintelligentbot\src\TalkToMyBot.sln and wait for it to load.
4. Right click the Solution and click Restore NuGet Packages.

Now let's install the LUIS package from NuGet:

1. Right-click on the ChatBot project and click Manage NuGet Packages.
2. Select the Browse tab and search for Microsoft.Bot.Builder.AI.Luis.
3. Click on the NuGet package, select the version 4.4 and click Install.

5.5 Add the LUIS Recognizer to your BOT

1. In Visual Studio, open the appsettings.json file.
2. Update the following LUIS configuration values:

- Replace <your_luis_app_id> and <your_luis_api_key> with the values you captured in Notepad earlier from Luis portal.

1. Open the Startup.cs file.
2. Add the following namespace at the top of the file:

```
3. using Microsoft.Bot.Builder.AI.Luis;
```

4. Find the comment // create luis recognizer around line 67 and replace it with the following code snippet:

```
5.  var luisApplication = new LuisApplication(
6.      Configuration["LuisAppId"],
7.      Configuration["LuisAPIKey"],
8.      "https://" + Configuration["LuisAPIHostName"]);
    services.AddSingleton(new LuisRecognizer(luisApplication));
```

5.6 Adjust your BOT to process LUIS results

1. Open EchoBot.cs.
2. Add the following namespace at the top of the file:

```
3. using Microsoft.Bot.Builder.AI.Luis;
```

4. Find the comment // Add Luis Recognizer and replace it with the following line:

```
5. protected LuisRecognizer _luis;
```

6. Replace the constructor header to be:

```
7. public EchoBot(EchoBotAccessors accessors, IOptions<MySettings> config,
   LuisRecognizer luisRecognizer)
```

8. Find the comment // Initialize the LUIS recognizer and replace it with the following line:

```
_luis = luisRecognizer;
```

10. Find the comment // Handle LUIS intents recognition and replace it with the following code snippet:

```
// Perform a call to LUIS to retrieve results for the current activity message.
var luisResults = await _luis.RecognizeAsync(turnContext, cancellationToken).ConfigureAwait(false);
var topScoringIntent = luisResults?.GetTopScoringIntent();
var topIntent = topScoringIntent.Value.score > 0.5 ? topScoringIntent.Value.intent : string.Empty;

// Your code goes here
```

> Note: The first step with LUIS is to extract the *intent* from the context. This is populated by the Recognizer.

17. Replace the recently added line // Your code goes here with the following code:

```
switch (topIntent)
{
    case "TodaysSpecialty":
        await turnContext.SendActivityAsync($"For today we have the following options: {string.Join(", ", BotConstants.Specialties)}");
        break;
    default:
        await turnContext.SendActivityAsync("Sorry, I didn't understand that.");
        break;
}
```

> Note: This switch will send the user's message to the right handler based on the LUIS intent name.

5.7 Test LUIS Configuration

1. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
2. Once the page has loaded in Microsoft Edge, copy the site host from the address bar to the clipboard (e.g. http://localhost:XXXX).
3. Return to the Bot Framework Emulator.
4. Click File -> New Bot Configuration.
5. Paste the URL that you copied from your browser and add /api/messages at the end of it. It should look something like: http://localhost:3978/api/messages.

   Note: The port might not be the same, but don't worry.
6. Click the Save and connect button.
7. Save the configuration to a file and continue to start a conversation.
8. Type what is the specialty for today? and press enter.
9. See the response, LUIS is processing our input and the BOT can handle it accordingly.
10. Stop debugging by clicking the stop button in VS toolbar.

5.8 Assess Your Questions

5.8.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, create a series of language-based sentences apt for the BOT to understand to accomplish a simple task of your choice.

5.9 Reference Links

- https://aischool.microsoft.com
- https://luis.ai/

Module 6: Adding Advanced Conversational Features to your BOT

Chapter 6: Overview of BOT Framework Emulator

Through this chapter we aim to make the students:

- Understand the working of the bot framework emulator.
- Successfully incorporate the features in the BOT.Test the incorporation and subsequent deployment of the features.

In this chapter, we will further delve into the ways of adding features to your BOT.

6.1 Add a Visual Response to your BOT: Carousel

1. Open EchoBot.cs.
2. Add the following method to the class around line 54:

```
3.  private async Task TodaysSpecialtiesHandlerAsync(ITurnContext context)
4.  {
5.      var actions = new[]
6.      {
7.          new CardAction(type: ActionTypes.ShowImage, title: "Carbonara", value:
    "Carbonara", image: $"{BotConstants.Site}/carbonara.jpg"),
8.          new CardAction(type: ActionTypes.ShowImage, title: "Pizza", value: "Pizza",
    image: $"{BotConstants.Site}/pizza.jpg"),
9.          new CardAction(type: ActionTypes.ShowImage, title: "Lasagna", value: "Lasagna",
    image: $"{BotConstants.Site}/lasagna.jpg"),
10.     };
11.
12.     var cards = actions
13.       .Select(x => new HeroCard
14.       {
15.           Images = new List<CardImage> { new CardImage(x.Image) },
16.           Buttons = new List<CardAction> { x },
17.       }.ToAttachment())
18.       .ToList();
19.     var activity = (Activity)MessageFactory.Carousel(cards, "For today we have:");
20.
```

```
21.     await context.SendActivityAsync(activity);
22. }
```

23. Modify the OnTurnAsync method, replace this line

```
24. await turnContext.SendActivityAsync($"For today we have the following options:
    {string.Join(", ", BotConstants.Specialties)}");
```

with

```
await TodaysSpecialtiesHandlerAsync(turnContext);
```

6.2 Test the Visual Response

1. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
2. Open the Models/BotConstants.cs file.
3. Check the Site constant with the URI of the site, if it's different than the URI in the browser stop the app and update this constant with the right port and run the app again.
4. Return to the Bot Framework Emulator.
5. Click the Restart conversation button to start a new conversation.
6. Type What is today's specialty? and press enter.
7. The BOT will display a set of food recommendations using images in a Carousel.
8. Stop debugging by clicking the stop button in VS toolbar.

6.3 Install and Set Up Dialogs Package

1. Right-click on the ChatBot project and click Manage NuGet Packages.
2. Select the Browse tab and search for Microsoft.Bot.Builder.Dialogs.
3. Click on the NuGet package, select version 4.4 and click Install.
4. Finish the installation process.

Configure the Dialog state:

1. Open EchoBotAccessors.cs.
2. Add the following namespace at the top of the file:

```
using Microsoft.Bot.Builder.Dialogs;
```

4. Add the following property at the end of the class:

```
public IStatePropertyAccessor<DialogState> ConversationDialogState { get; set; }
```

6. Open the ReservationDialog.cs file inside the Dialogs folder.
7. Add the following namespace at the top of the file:

```
using Microsoft.Bot.Builder.Dialogs;
```

9. Update the class definition to be:

```
public class ReservationDialog : ComponentDialog
```

11. Update the constructor definition to be:

```
public ReservationDialog(
    IStatePropertyAccessor<ReservationData> userProfileStateAccessor,
    TextToSpeechService ttsService)
    : base(nameof(ReservationDialog))
```

16. Add the following method:

```
private async Task<DialogTurnResult> InitializeStateStepAsync(WaterfallStepContext
    stepContext, CancellationToken cancellationToken)
{
    var state = await UserProfileAccessor.GetAsync(stepContext.Context, () => null);
```

```
20.    if (state == null)
21.    {
22.        var reservationDataOpt = stepContext.Options as ReservationData;
23.        if (reservationDataOpt != null)
24.        {
25.            await UserProfileAccessor.SetAsync(stepContext.Context, reservationDataOpt);
26.        }
27.        else
28.        {
29.            await UserProfileAccessor.SetAsync(stepContext.Context, new
       ReservationData());
30.        }
31.    }
32.
33.    return await stepContext.NextAsync();
34. }
```

Note:* This method will be the first step of the dialog and is in charge to initialise the state if it's not already.

35. Open Startup.cs.
36. Add the following namespace at the top of the file:

```
37. using Microsoft.Bot.Builder.Dialogs;
```

38. Find the comment Initialize Dialog State and replace that line with the following code snippet:

```
39. ConversationDialogState =
    conversationState.CreateProperty<DialogState>("DialogState"),
40. ReservationState = userState.CreateProperty<ReservationData>("ReservationState"),
```

6.4 Set Up the Conversation Flow

1. Open EchoBot.cs.
2. Add the following namespace at the top of the file:

```
using Microsoft.Bot.Builder.Dialogs;
```

3. Find the comment // Add Dialogs set and replace it with the following line:

```
private readonly DialogSet _dialogs;
```

4. Find the comment // Initialize the dialogs and replace it with the following line:

```
_dialogs = new DialogSet(_accessors.ConversationDialogState);
```

5. Find the comment // Register the dialog and replace it with the following line:

```
_dialogs.Add(new ReservationDialog(_accessors.ReservationState, null));
```

6. Modify the OnTurnAsync method by adding the following code snippet before the line if (!turnContext.Responded):

```
var dialogContext = await _dialogs.CreateContextAsync(turnContext,
cancellationToken);
var dialogResult = await dialogContext.ContinueDialogAsync(cancellationToken);
```

7. Replace the content of if (!turnContext.Responded) with the following code snippet:

```
switch (dialogResult.Status)
{
    case DialogTurnStatus.Empty:
        // Your code goes here

        break;
    case DialogTurnStatus.Waiting:
        // The active dialog is waiting for a response from the user, so do nothing.
        break;
    case DialogTurnStatus.Complete:
        await dialogContext.EndDialogAsync();
        break;
```

```
    default:
        await dialogContext.CancelAllDialogsAsync();
        break;
}
```

Note: This allows to handle the result depending on the dialog status.

30. Replace the recently added line `// Your code goes here` with the following code:

```
// Check LUIS model
var luisResults = await _luis.RecognizeAsync(turnContext,
cancellationToken).ConfigureAwait(false);
var topScoringIntent = luisResults?.GetTopScoringIntent();
var topIntent = topScoringIntent.Value.score > 0.5 ? topScoringIntent.Value.intent :
string.Empty;
switch (topIntent)
{
    case "TodaysSpecialty":
        await TodaysSpecialtiesHandlerAsync(turnContext);
        break;
    case "ReserveTable":
        var amountPeople = luisResults.Entities["AmountPeople"] != null ?
(string)luisResults.Entities["AmountPeople"]?.First : null;
        var time = GetTimeValueFromResult(luisResults);
        await ReservationHandlerAsync(dialogContext, amountPeople, time,
cancellationToken);
        break;
    default:
        await turnContext.SendActivityAsync("Sorry, I didn't understand that.");
        break;
}
```

Note: The missed methods will be added later.

49. Add the following method at the end of the class to extract the reservation date:

```
50. private string GetTimeValueFromResult(RecognizerResult result)
51. {
52.     var timex = (string)result.Entities["datetime"]?.First["timex"].First;
53.     if (timex != null)
54.     {
55.         timex = timex.Contains(":") ? timex : $"{timex}:00";
56.         return DateTime.Parse(timex).ToString("MMMM dd \\a\\t HH:mm tt");
57.     }
58.
59.     return null;
60. }
```

Note: The time returned by LUIS contains the datetime as a string in a property called 'timex'.

61. Add the ReservationHandlerAsync method to start the conversation flow for the table reservation intent:

```
62. private async Task ReservationHandlerAsync(DialogContext dialogContext, string
    amountPeople, string time, CancellationToken cancellationToken)
63. {
64.     var state = await _accessors.ReservationState.GetAsync(dialogContext.Context, ()
    => new ReservationData(), cancellationToken);
65.     state.AmountPeople = amountPeople;
66.     state.Time = time;
67.     await dialogContext.BeginDialogAsync(nameof(ReservationDialog));
68. }
```

Note: The method invokes the ReservationDialog pushed in the constructor.

69. Put a breakpoint on the await dialogContext.BeginDialogAsync(GatherInfo); line in the ReservationHandlerAsync.
70. Find the comment // Save states in the accessor in the OnTurnAsync method and add the following code snippet after that:

```
71. // Get the conversation state from the turn context.
```

```
var state = await _accessors.ReservationState.GetAsync(turnContext, () => new
ReservationData());

// Set the property using the accessor.
await _accessors.ReservationState.SetAsync(turnContext, state);

// Save the new state into the conversation state.
await _accessors.ConversationState.SaveChangesAsync(turnContext);
await _accessors.UserState.SaveChangesAsync(turnContext);
```

80. Open the ReservationDialog.cs file inside the Dialogs folder.
81. Find the comment // Add control flow dialogs and add after the following code snippet:

```
AddDialog(new WaterfallDialog(ProfileDialog, waterfallSteps));
AddDialog(new TextPrompt(TimePrompt));
AddDialog(new TextPrompt(AmountPeoplePrompt, AmountPeopleValidatorAsync));
AddDialog(new TextPrompt(NamePrompt));
AddDialog(new ConfirmPrompt(ConfirmationPrompt));
```

> Note: This will set up the conversation flow passing the Luis results between the steps.

87. Add the following method to request the reservation date if it wasn't provided in the initial utterance:

```
private async Task<DialogTurnResult> TimeStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
{
    var state = await UserProfileAccessor.GetAsync(stepContext.Context);

    if (string.IsNullOrEmpty(state.Time))
    {
        var msg = "When do you need the reservation?";
        var opts = new PromptOptions
```

```
96.       {
97.           Prompt = new Activity
98.           {
99.               Type = ActivityTypes.Message,
100.                  Text = msg,
101.                  // Add the message to speak
102.              },
103.          };
104.          return await stepContext.PromptAsync(TimePrompt, opts);
105.      }
106.      else
107.      {
108.          return await stepContext.NextAsync();
109.      }
110.  }
```

111. Add the following method to request the amount of people in the reservation if it wasn't provided in the initial utterance:

```
112. private async Task<DialogTurnResult>
     AmountPeopleStepAsync(WaterfallStepContext stepContext, CancellationToken
     cancellationToken)
113. {
114.     var state = await UserProfileAccessor.GetAsync(stepContext.Context);
115.
116.     if (stepContext.Result != null)
117.     {
118.         var time = stepContext.Result as string;
119.         state.Time = time;
120.     }
121.
122.     if (state.AmountPeople == null)
123.     {
```

```
124.         var msg = "How many people will you need the reservation for?";
125.         var opts = new PromptOptions
126.         {
127.             Prompt = new Activity
128.             {
129.                 Type = ActivityTypes.Message,
130.                 Text = msg,
131.                 // Add the message to speak
132.             },
133.         };
134.         return await stepContext.PromptAsync(AmountPeoplePrompt, opts);
135.     }
136.     else
137.     {
138.         return await stepContext.NextAsync();
139.     }
140.  }
```

Note: Notice how we extract the response from the previous step and update the state in each step.

141. Add the following method to request the name on the reservation:

```
142.  private async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext
      stepContext, CancellationToken cancellationToken)
143. {
144.     var state = await UserProfileAccessor.GetAsync(stepContext.Context);
145.
146.     if (stepContext.Result != null)
147.     {
148.         state.AmountPeople = stepContext.Result as string;
149.     }
150.
151.     if (state.FullName == null)
```

```
    {
        var msg = "And the name on the reservation?";
        var opts = new PromptOptions
        {
            Prompt = new Activity
            {
                Type = ActivityTypes.Message,
                Text = msg,
                // Add the message to speak
            },
        };
        return await stepContext.PromptAsync(NamePrompt, opts);
    }
    else
    {
        return await stepContext.NextAsync();
    }
}
```

Add the following methods to request a confirmation and finalise the reservation:

```
private async Task<DialogTurnResult> ConfirmationStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var state = await UserProfileAccessor.GetAsync(stepContext.Context);

    if (stepContext.Result != null)
    {
        state.FullName = stepContext.Result as string;
    }

    if (state.Confirmed == null)
```

```
    {
        var msg = $"Ok. Let me confirm the information: This is a reservation for {state.Time} for {state.AmountPeople} people. Is that correct?";
        var retryMsg = "Please confirm, say 'yes' or 'no' or something like that.";

        var opts = new PromptOptions
        {
            Prompt = new Activity
            {
                Type = ActivityTypes.Message,
                Text = msg,
                // Add the message to speak
            },
            RetryPrompt = new Activity
            {
                Type = ActivityTypes.Message,
                Text = retryMsg,
                // Add the retry message to speak
            },
        };
        return await stepContext.PromptAsync(ConfirmationPrompt, opts);
    }
    else
    {
        return await stepContext.NextAsync();
    }
}
```

Add the following method to verify the confirmation response and finish the flow.

```
private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
```

```
{
    var state = await UserProfileAccessor.GetAsync(stepContext.Context);

    if (stepContext.Result != null)
    {
        var confirmation = (bool)stepContext.Result;
        string msg = null;
        if (confirmation)
        {
            msg = $"Great, we will be expecting you this {state.Time}. Thanks for your reservation {state.FirstName}!";
        }
        else
        {
            msg = "Thanks for using the Contoso Assistance. See you soon!";
        }

        await stepContext.Context.SendActivityAsync(msg);
    }

    return await stepContext.EndDialogAsync();
}
```

Add the following method to validate that the number of people specified is a number:

```
private async Task<bool> AmountPeopleValidatorAsync(PromptValidatorContext<string> promptContext, CancellationToken cancellationToken)
{
    var value = promptContext.Recognized.Value?.Trim() ?? string.Empty;

    if (!int.TryParse(value, out int numberPeople))
```

```
    {
        var msg = "The amount of people should be a number.";
        await promptContext.Context.SendActivityAsync(msg)
          .ConfigureAwait(false);
        return false;
    }
    else
    {
        promptContext.Recognized.Value = value;
        return true;
    }
}
```

Find the // Add control flow steps comment and add after the following code snippet:

```
var waterfallSteps = new WaterfallStep[]
{
    InitializeStateStepAsync,
    TimeStepAsync,
    AmountPeopleStepAsync,
    NameStepAsync,
    ConfirmationStepAsync,
    FinalStepAsync,
};
```

6.5 Test the Conversation Flow

1. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
2. Return to the Bot Framework Emulator.
3. Click the Restart conversation button to start a new conversation.
4. Type I need a dinner reservation for tomorrow at 7:30 pm and press enter.

5. Return to Visual Studio and wait for the breakpoint to be hit.
6. Mouse over the amountPeople and time variables to inspect their content.

> Note: Only the time will have a value, as this is the only piece of information provided in our initial utterance.

7. Press Continue in the toolbar.
8. Return to the Bot Framework Emulator.
9. The BOT will request the amount of people in the reservation. Type a abc and press enter.

> Note: Notice the validation working, in this case the BOT will request the amount of people again.

10. Type a 2 and press enter.
11. The BOT will request the name of the reservation. Type a Jane Olson and press enter.
12. The BOT will request a confirmation. Type yes and press enter.

> Note: At this point the conversation flow will be finished.

13. Stop debugging by clicking the stop button in VS toolbar.
14. Remove the breakpoint previously added.

6.5 Assess Your Knowledge

6.5.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, deploy the Bot Framework Emulator features for the activity/task done in the previous chapter.

6.6 Reference Links

- https://www.visualstudio.com
- https://aischool.microsoft.com

Module 7: Adding Knowledge to your BOT Using QnA Maker

Chapter 7: Overview of QnA Maker

Through this chapter we aim to make the students:

- Understand QnA Maker and it features.

7.1 Introduction to QnA Maker

Let's add QnA Maker knowledge service to help your BOT to retrieve answers easily, this service makes possible to respond to user's questions in a more natural, conversational way using a simple Question and Answer format.

QnA Maker enables you to power a question and answer service from your semi-structured content like FAQ (Frequently Asked Questions) documents or URLs and product manuals. You can build a model of questions and answers that is flexible to user queries, providing knowledge responses for your BOTs to use in a natural, conversational way.

7.2 Set Up QnA Maker

Create your QnA Maker Azure resource:

1. Return to the Azure Portal(portal.azure.com).
2. Click Create Resource [+] from the left menu and search for QnA Maker.
3. Select the first result and then click the Create button.
4. Provide the required information:

    - Name: chatbot-qna-<your initials>
    - Management Pricing tier: F0 (3 Calls per second)
    - Use existing resource group: ttmb-lab-<your initials>
    - Search pricing tier: F (3 Indexes, 10K Documents)
    - Search Location: West US
    - App name: build-qna-<your initials>
    - Website Location: West US
    - Application Insights Location: West US 2

5. Click Create to deploy the service. This step might take a few moments.
6. Once the deployment is complete, you will see a Deployment succeeded notification.
7. Go to All Resources in the left pane and search for the new resource (chatbot-qna-<your initials>).
8. Click on the resource.
9. Go to the Keys page.
10. Copy the Key 1 value into Notepad.

> Note: We'll need this key later on.

7.3 Set Up your QnA Maker Instance

1. Log into the QnA Maker portal (qnamaker.ai) using your Azure credentials.
2. Create a knowledge base:
    - Click on Create a knowledge base.
    - Scroll down to Step 2: Connect your QnA service to your KB.
    - Select the previously created Azure service.
    - Scroll down to Step 3: Name your KB.
    - Enter the name of the KB: qna-<your initials>
    - Scroll down to Step 4: Populate your KB.
    - Click Add File and select the knowledge base file provided: resources/qna-ttmb-KB.tsv
3. Click Create your KB and wait for the new instance to load.

> Note: You will be redirected to the QnA dashboard and it will display the questions in you knowledge base, these are populated from the previously loaded file

4. Click Save and train. This should take a minute.
5. Click Publish to start the publishing process and then Publish again to confirm.
6. From the sample HTTP request, copy the following values to a Notepad:
    - Host url.

- EndpointKey from the Authorization header
- KnowledgeBaseId from the POST

  Uri: knowledgebases/<knowledgeBaseId>/generateAnswer

7.4 Install QnA Maker Package

1. In Visual Studio, right click on the ChatBot project and click Manage NuGet Packages.
2. Select the Browse tab and search for Microsoft.Bot.Builder.AI.QnA.
3. Select the NuGet package, select version 4.4 and click Install.

7.5 Assess Your Knowledge

7.5.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, create a knowledge base for the activity/task done in the previous chapter.

7.6 Reference Links

- https://www.visualstudio.com
- https://aischool.microsoft.com

Chapter 8: Integration of QnA Maker in BOT

Through this chapter we aim to make the students:

- Understand integration and deployment of QnA Maker in the Bot Emulator.

Notes for the Faculty

The faculty is to guide the students via the given steps towards adding the QnA maker to the Bot Emulator.

8.1 Add QnA Maker to the BOT

The new Bot Builder SDK V4 provides a middleware to easily integrate QnA Maker in your BOT. We will use this service to connect our BOT to the previously created Knowledge Base.

1. Return to Visual Studio.
2. Open appsettings.json and add the QnA Maker settings:

```
"QnAKbId": "<your knowledge base id>",
"QnAEndpointKey": "<your qna maker subscription key>",
"QnAHostname": "<your qna maker url>"
```

Note: Make sure you update the host, endpoint key, and knowledge base ID.

6. Open Startup.cs.
7. Add the following namespaces at the top of the file:

```
using Microsoft.Bot.Builder.AI.QnA;
```

9. Look for the line Add QnA Maker here add the following code before it:

```
// We add the the QnA service to the connected services.
// Create and register a QnA service and knowledgebase
services.AddSingleton(sp =>
{
    return new QnAMaker(
        new QnAMakerEndpoint
        {
            EndpointKey = Configuration["QnAEndpointKey"],
```

```
18.          Host = Configuration["QnAHostname"],
19.          KnowledgeBaseId = Configuration["QnAKbId"],
20.      },
21.      new QnAMakerOptions
22.      {
23.          ScoreThreshold = 0.9f,
24.          Top = 1,
25.      });
26. });
```

27. Open EchoBot.cs.
28. Add the following import at the top of the class:

```
29. using Microsoft.Bot.Builder.AI.QnA;
```

30. Find the comment // Add QnAMaker and add after that the following code snippet:

```
31. private QnAMaker QnA { get; } = null;
```

32. Change the definition of the constructor from:

```
33. public EchoBot(EchoBotAccessors accessors, IOptions<MySettings> config,
    LuisRecognizer luisRecognizer)
```

to

```cs
public EchoBot(EchoBotAccessors accessors, IOptions<MySettings> config, LuisRecognizer luisRecognizer, QnAMaker qna)
```

1. Find the line _accessors = accessors ?? throw new ArgumentNullException(nameof(accessors)); and add the following line after:

```
QnA = qna ?? throw new ArgumentNullException(nameof(qna));
```

1. In the method OnTurnAsync find the line await turnContext.SendActivityAsync("Sorry, I didn't understand that."); and replace it with the following code snippet:

2.
```
var answers = await this.QnA.GetAnswersAsync(turnContext);
```

```
3.
4. if (answers is null || answers.Count() == 0)
5. {
6.     await turnContext.SendActivityAsync("Sorry, I didn't understand that.");
7. }
8. else if (answers.Any())
9. {
10.    // If the service produced one or more answers, send the first one.
11.    await turnContext.SendActivityAsync(answers[0].Answer);
12. }
```

8.2 Test QnA Maker in Bot Emulator

1. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
2. Return to the Bot Framework Emulator.
3. Click the Restart conversation button to start a new conversation.
4. Type where are you located? and press enter.
5. Check the response from the BOT using QnA Maker now.
6. Return to Visual Studio and stop debugging by clicking the stop button in the toolbar.

8.3 Assess Your Knowledge

8.3.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, integrate the QnA maker to the Bot Emulator for the activity/task done in the previous chapter.

8.4 Reference Links

- https://aischool.microsoft.com

Module 8: Implementing Personality Chat

Chapter 9: Overview of Personality Chat

Through this chapter we aim to make the students:

- Understand the concept of personality chat in a BOT.
- Understand the integration and deployment of the same in the Bot Emulator.

9.1 Overview

While infusing 'brains' to BOTs, often comes the need to perform intelligent conversations with sometimes an additional layer of personality to make interactions more human. Part of Cognitive Services Labs, Project Personality Chat helps make intelligent agents more complete and conversational by handling common small talk in a consistent tone and reducing fallback responses, such as “I don’t understand.” Developers can give their agents a personality by choosing their preferred persona, from professional to humorous. Project Personality Chat uses a chat classifier to identify common small talk intents and Deep Neural Networks (DNNs) to generate contextual responses. As a complement to the API, a customisable editorial library for common small talk queries can be easily integrated using the Azure Bot Service SDK saving developers the time and cost of writing them from scratch.

9.2 Add Project Personality Chat to the BOT

The Bot Builder SDK enables chitchat capabilities by using the PersonalityChat middleware. The code provided, already includes the Personality Chat NuGet package `Microsoft.Bot.Builder.PersonalityChat`, all we have to do is configure the middleware.

1. Return to Visual Studio.
2. Open Startup.cs.
3. Find the comment // Add the personality chat middleware and add after the following code snippet:

```
var personalityChatOptions = new PersonalityChatMiddlewareOptions(
  respondOnlyIfChat: true,
  scoreThreshold: 0.5F,
  botPersona: PersonalityChatPersona.Humorous);
services.AddSingleton(new PersonalityChatMiddleware(personalityChatOptions));
```

Note: For this sample we will use the default "Humorous" personality, other options are: Professional and Friendly (default).

9. Open AdapterWithErrorHandler.cs.
10. Replace the constructor header to be:

```
public AdapterWithErrorHandler(ICredentialProvider credentialProvider, ILogger<BotFrameworkHttpAdapter> logger, PersonalityChatMiddleware personalityChatMiddleware, ConversationState conversationState = null)
    : base(credentialProvider)
```

13. Find the comment `// Use personality chat middleware` and add the following code snippet:

```
if (personalityChatMiddleware == null)
{
    throw new NullReferenceException(nameof(personalityChatMiddleware));
}

Use(personalityChatMiddleware);
```

9.3 Test Personality Chat in Bot Emulator

1. Run the app by clicking on the IIS Express button in Visual Studio (with the green play icon).
2. Return to the Bot Framework Emulator.
3. Click the Restart conversation button to start a new conversation.
4. Type how old are you? and press enter. See the response from your BOT.
5. Return to Visual Studio and stop debugging by clicking the stop button in the toolbar.
6. Try changing the personality from Startup.cs to be something like botPersona: PersonalityChatPersona.Professional and see how the BOT response changes accordingly to fit the selected personality.

Note: Review the image below to see which utterances you can try and what output to expect in each case.

9.4 Assess Your Knowledge

9.4.1 Practical Assignment and Lab Work

In groups of three or more, as decided by the faculty, design the personality chat and incorporate the same for the activity/task done in the previous chapter.

9.5 Reference Links

- https://aischool.microsoft.com

Chapter 10: Adding Speech Support

- Through this chapter we aim to make the students:Integrate the component of speech support elements to the BOT.
- Deploy the same using Azure via Visual Studio.

In this section, we will enable speech in Web Chat to recognise speech and send the transcript to the BOT. We will also use Text-to-Speech from Speech Service to generate SSML and send audio back from the Web Chat.

10.1 Overview

One of the key cognitive capabilities to make your BOT more natural is for the system to understand verbal requests, through speech and to respond in kind with speech. Microsoft Speech services make it possible to understand many requests, even in another language.

The Speech-to-Text API helps developers to easily add real-time, speech-enabled features to their applications, enabling a wide range of uses cases like voice commands, real-time transcriptions or call center log analysis. The general model of Speech Service converts speech to text for a variety of languages, whether the audio is coming from a microphone, other real-time audio sources or an audio file.

10.2 Create a Speech Subscription

1. Return to the Azure Portal (portal.azure.com).
2. Click Create Resource [+] from the left menu and search for Speech.
3. Select the *Speech* result and then click the Create button.
4. Provide the required information:
    - App name: speech-<your_initials>.
    - Location: East US.
    - Pricing tier: S0.
    - Use existing resource group: ttmb-lab-<your initials>.

> Note: Make sure to select East US for the location as the neural voice we will use ahead on the lab is available only in a limited set of regions. Here you can find more information about the neural voices regions.

5. Click Create. This step might take a few seconds.
6. Once the deployment is complete, you will see a Deployment succeeded notification.
7. Go to All Resources in the left pane and search for the new resource (speech-<your initials>).
8. Click on the resource.
9. Go to the Keys page.
10. Copy the Key 1 value into Notepad.

> Note: We'll need this key later on.

10.3 Set Up Web Chat

Speech is available as a component called `SpeechRecognizer` in the Web Chat control. The Speech Recognizer generates a transcript from audio; this will allow us to get the text from the user speech and then send that text to the BOT. Before we can set up speech capabilities, we need to configure the Web Chat control.

1. From the Azure Portal, click on the Channels option of your Web App Bot.
2. Under Add a featured channel, select Configure Direct Line Channel.
3. In the Secret Keys section, click the Show toggle button to display the password.
4. Copy the first password to clipboard.
5. Return to Visual Studio.
6. Open the default.html file in the wwwroot folder.

> Note: The Web Chat control has already been imported into the page, we just need to configure it. This Web Chat was modified to use the speech websockets instead of the general Bing service.

7. Replace the direct-line-secret from line 181 with the value on clipboard.
8. Replace the Speech Subscription key <your-speech-subscription-key> from line 97 with the value previously obtained from Azure.

10.4 Add Text to Speech to BOT

For this scenario, we will generate the audio SSML in the backend (our BOT code) using Neural TTS, a text-to-speech synthesis with a production system that uses deep neural networks to make the voices of computers nearly indistinguishable from recordings of people. We'll use one of the pre-built neural voices in English and play the audio by using the Speech synthesizer component from the Web Chat.

1. Open EchoBot.cs.
2. Find the comment // Add text to speech service and replace it with:

3. `private readonly TextToSpeechService _ttsService;`
4. Find the comment // Initialize the TTSS and replace it with:

5. `_ttsService = new TextToSpeechService(config.Value.VoiceFontName, config.Value.VoiceFontLanguage);`

   Note: The TextToSpeechService selects the JessaNeural voice for the en-US language, hereyou can find more information about Neural voices.
6. Find the line in the constructor _dialogs.Add(new ReservationDialog(_accessors.ReservationState, null)); and replace the null at the end with _ttsService to include the service to the dialog.
7. Modify the *OnTurnAsync* method. Look for the welcome message "Hi! I'm a restaurant assistant bot. I can help you with your reservation. and replace the next line (await turnContext.SendActivityAsync(msg);) to include the audio SSML in the response:

8. `await turnContext.SendActivityAsync(msg, _ttsService.GenerateSsml(msg, BotConstants.EnglishLanguage));`

9. Open the ReservationDialog in the Dialogs folder.
10. Find the // Add the message to speak comment in the TimeStepAsync method and replace it with the following line:

11. `Speak = _ttsService.GenerateSsml(msg, BotConstants.EnglishLanguage),`

    Note: Notice that we only modified the Speak option and set the SSML to the response activity. As an alternative, we could just send the message and the Bot

Framework Web Chat will generate the SSML for you, but for this lab we will send the SSML from the backend.

12. Repeat the previous step to add the speak property in the AmountPeopleStepAsync and NameStepAsync methods.
13. Modify the ConfirmationStepAsync method. Replace the opts variable with the following code snippet:

```
var opts = new PromptOptions
{
    Prompt = new Activity
    {
        Type = ActivityTypes.Message,
        Text = msg,
    Speak = _ttsService.GenerateSsml(msg, BotConstants.EnglishLanguage),
    },
    RetryPrompt = new Activity
    {
        Type = ActivityTypes.Message,
        Text = retryMsg,
    Speak = _ttsService.GenerateSsml(msg, BotConstants.EnglishLanguage),
    },
};
```

29. Modify the FinalStepAsync method. Replace the await stepContext.Context.SendActivityAsync(msg); line with the following code:

```
await stepContext.Context.SendActivityAsync(msg, _ttsService.GenerateSsml(msg,
BotConstants.EnglishLanguage));
```

31. Modify the AmountPeopleValidatorAsync method. Replace the line await promptContext.Context.SendActivityAsync(msg) with the following code:

```
await promptContext.Context.SendActivityAsync(msg, _ttsService.GenerateSsml(msg,
BotConstants.EnglishLanguage))
```

10.5 Deploy to Azure from Visual Studio

1. Return to the Azure Portal(portal.azure.com).
2. Search for the Web App Bot resource in your group.
3. Click the Configuration option of the menu in the left under the App Service Settings category.
4. Click the Edit icon for the settings MicrosoftAppId and MicrosoftAppPassword and copy the values for both in notepad.
5. Delete all the following settings if they are set up:

    - LuisAPIHostName
    - LuisAPIKey
    - LuisAppId

    Note: This is because we will use the settings from the project app settings file.
6. If you delete at least one of the settings, click the Save button at the top.
7. Return to Visual Studio.
8. Open the appsettings.json file.
9. Replace the values of MicrosoftAppId and MicrosoftAppPassword with the values you got from Azure.
10. Click on the current connected account in the top right corner of Visual Studio.
11. Click on Account Settings....
12. Click on the Sign out button.
13. Click on the Sign in button.
14. Login with the same credentials as you used for Azure.

    Note: This will connect Visual Studio to your Azure subscription.
15. Click Close.
16. Right-click the ChatBot project.
17. Click Publish.
18. Mark the option Select Existing.
19. Click Publish.

20. Select the bot Web App previously created under the group ttmb-lab-.
21. Click OK and wait for the deployment to complete. This step might take a few minutes.

10.6 Test in Web Chat

1. Open your chatbot in a browser: https://chatbot-<your initials>.azurewebsites.net/.
2. Click on the Speech option.

> Note: If something is not working make sure that you are using a secure connection by using https:// in your URL.

3. Click the microphone icon once and say I need a table reservation.
4. Wait for the BOT to reply, you should get an audio response back.
5. Finish the conversation flow using audio.

Chapter 11: Overview of Custom Speech Model

- Through this chapter we aim to make the students:Integrate the components of speech customisation in to the BOT.

This section will show you how to get started with using the Custom Speech Service to improve the accuracy of speech-to-text transcription in your application. We'll use a Language Model to improve the output of food domain-specific language.

11.1 Overview

In some cases, it can become a challenge to handle voice-enabled conversations in a complex or noisy environment. With the custom speech models, developers can tailor speech recognition models to overcome a variety of common comprehension challenges. Customise language models to adapt to users' speaking styles, dialects, expressions or unique vocabulary like place names, products or industry-specific terms. Tailor acoustic models to accommodate specific background noise, accents and voice patterns depending on your scenario.

Using custom speech might also help to overcome speech impairments and support linguistic diversity by building your own speech models. It's important to explicitly include diversity in the design of your feedback mechanisms to encourage representation and avoid alienation of groups that would otherwise not be well represented. Take some time to read the Fairness and Diversity section of our Responsible Conversational AI to get more information about Fairness and Diversity guidelines via the following link: https://aischool.microsoft.com/en-us/responsible-ai/learning-paths/responsible-conversational-ai/building-a-transparent-and-trustworthy-bot/fairness-and-diversity .

11.2 Speech Customisation: Using Custom Speech

In order to compare the performance of the custom speech service, we'll create a custom speech model to process more specific language. This requires to link your Azure Speech subscription to the Custom Speech service using the CRIS Portal.

1. Log into the CRIS Dashboard (www.cris.ai).
2. Click on your user account in the right side of the top ribbon and click on All Subscriptions in the drop-down menu.
3. Click Connect existing subscription and provide the requested information:
    - Subscription Key: *paste speech subscription value previously obtained from Azure Portal*
4. Click Add. It will display a page with the result of your subscription validation.

Custom Speech models are a great option to meet the hearing and speech needs for users with disabilities or impairments. Making your BOTs accessible implies that they will be available to a wider audience. Read the Make Your Bot Accessible section of Responsible Conversational AI learning path to learn more.

11.3 Customise Language Model and Pronunciation

Building a custom language model allows us to improve the vocabulary of our speech model and specify the pronunciation of specific words.

1. Open Downloads/buildanintelligentbot/resources/custom_speech_language_ds.txt with a Text Editor and review its contents. This file contains text examples of queries and utterances expected from users.
2. Click the Custom Speech from the top menu and select Adaptation Data.
3. Click Import next to Language Datasets and provide the requested information:
    - Name: Food Language Model.
    - Language data file (.txt): navigate and select buildanintelligentbot/resources/custom_speech_language_ds.txt from the Downloadsfolder.
4. Click Import. It will display a table with your new dataset. Wait for the Status to change to Succeeded.
5. Click Import next to Pronunciation Datasets and provide the requested information:
    - Name: Custom Pronunciation Data.

- Language data file (.txt): navigate and select buildanintelligentbot/resources/custom_speech_pronunciation_ds.txt from the Downloads folder.

6. Click Import. It will display a table with your new dataset. Wait for the Status to change to Succeeded.
7. Click on Language Models from the top menu.
8. Click Create New and provide the requested information:
    - Name: Food Language Model.
    - Locale: en-US.
    - Scenario: v4.5 Unified.
    - Language Data: auto populated with the sample data previously created.
    - Pronunciation Data: Custom Pronunciation Data.
9. Click Create. It will display a table with your new model. This can take several minutes to complete (between 20 - 30 minutes).

11.4 Create Custom Speech-to-Text Endpoint

Now that our model has finished building, we can quickly turn it into a web service endpoint that we can easily access from a mobile phone, tablet or PC app.

1. Click the Endpoints option from the top menu.
2. Click Create New and provide the requested information:
    - Name: cris-ttmb-<your initials>.
    - Scenario: v4.5 Unified.
    - Language Model: the model previously created should appear selected.
    - Accept terms and conditions
3. Click Create. It will display a table with your new deployment. Wait for the Status to change to Succeeded.
4. Click Details and scroll down to the Endpoints section.

> Note: This section will display your custom endpoint URLs. You can choose either HTTP REST API or WebSocket with the Speech Protocol/JavaScript WebSocket API.

5. Copy the displayed Endpoint ID into Notepad as you'll need it later.

> Note: Notice the WebSocket API endpoint URL (the one marked for audio up to 15 seconds). The lab Web Chat uses that endpoint in the background to execute the custom speech to text transcription.

11.5 Modify Chatbot to Support Custom Speech

We can easily access our new model by just adding the Custom Speech endpoint ID to the Web Chat. We will use the websockets to stream the audio and get the transcript back.

1. Return to Visual Studio.
2. Open default.html in the wwwroot folder.
3. Replace the Custom Speech Endpoint Id <custom-speech-endpoint-id> from line 98 with the value previously obtained from Cris dashboard.

> Note: The pre-built chatbot included in this tutorial already provides support for Custom Speech using websockets, we just have to add the endpoint ID configuration to the web page. Notice how we can access all the speech services using the same Speech subscription key.

11.6 Test Custom Speech in Web Chat

In order to compare the default Bing Speech service with the Custom Speech model that we configured, we will be using the 'risotto' and 'calzone' words that are part of the pronunciations dataset. The pronunciation of these words varies from Italian to English, carefully listen to the output audios and notice the difference, as we will be using an Italian accent to try this scenario.

Follow the next steps to listen the words pronunciation:

1. Open your browser and go to https://www.bing.com/Translator.
2. Type in calzone and select Italian from the list of languages.

3. Click on the listen icon.
4. In the second pane from the translator select English from the list of languages.
5. Click the listen icon and compare the pronunciation between English and Italian for this word.
6. Repeat the previous steps for risotto.

Deploy and test custom speech:

1. Go to Visual Studio and Right-click the ChatBot project.
2. Click Publish.
3. Click Publish again and wait for the deployment to complete. This step might take a few minutes.
4. Open your chatbot in a browser: https://chatbot-<your initials>.azurewebsites.net/.
5. Click the Speech option and try the following utterances using an *Italian* accent for the highlighted words:

    - Click the microphone icon once and say *Do you have risotto?*
    - Click the microphone icon once and say *Do you have calzone?*

    Note: The default speech service won't be able to understand these utterances and will reply with: Sorry, I didn't understand that.
6. Click the Custom Speech option and try the same utterances again:

    - Click the microphone icon once and say *Do you have risotto?*
    - The BOT will reply with a response from our Knowledge Base: *We have a large selection of risottos in our menu, our recommendation is risotto ai funghi.*
    - Click the microphone icon once and say *Do you have calzone?*
    - The BOT will reply with a response from our Knowledge Base: *We sell calzones only Fridays.*

    Note: Notice that Custom Speech is able to pick up the food options that we specified using a different pronunciation.

Chapter 12: Adding Speech Translation to your BOT

- Through this chapter we aim to make the students:Integrate the components of speech translation in to the BOT.

12.1 Overview

Machine translation systems are applications or online services that use machine learning technologies to translate large amounts of text from and to any of their supported languages. The service translates a 'source' text from one language to a different 'target' language. Although the concepts behind machine translation technology and the interfaces to use it are relatively simple, the science and technologies behind it are extremely complex and bring together several leading-edge technologies, in particular, deep learning (artificial intelligence), big data, linguistics, cloud computing and web APIs.

Speech Translation enables real-time speech translation capabilities with models based on neural machine translation (NMT) technologies. Three elements of the Speech Translation pipeline can now be customised: speech recognition, text to speech and machine translation.

With Custom Translator, enterprises, app developers and language service providers can build neural translation systems that understand the terminology used in their own business and industry. The customised translation system will then seamlessly integrate into existing applications, workflows and websites. Custom Translator allows users to customise Microsoft Translator's advanced neural machine translation when translating text using the Microsoft Translator Text API and speech translation using the preview of Cognitive Services Speech.

By using the translator, we can add support for linguistic diversity and enable communication through different languages. It's important to explicitly include diversity in the design of your feedback mechanisms to encourage representation and avoid alienation of groups that would otherwise not be well represented. For more information, read the Fairness and Diversity at https://aischool.microsoft.com/en-us/responsible-ai/learning-paths/responsible-conversational-ai/building-a-transparent-and-trustworthy-bot/fairness-and-diversity section of our Responsible Conversational AI learning path.

12.2 Create a Translator Text Subscription

1. Return to the Azure Portal (portal.azure.com).
2. Click Create Resource [+] from the left menu and search for Translator Text.
3. Select the *Translator Text* result and then click the Create button.
4. Provide the required information:
    - Name: translator-text-<your_initials>.
    - Subscritpion: your azure subscription.
    - Pricing tier: F0 (2M Up to 2M characters translated).
    - Use existing resource group: ttmb-lab-<your initials>.
5. Click Create. This step might take a few seconds.
6. Once the deployment is complete, you will see a Deployment succeeded notification.
7. Go to All Resources in the left pane and search for the new resource (translator-text-<your initials>).
8. Click on the resource.
9. Go to the Keys page.
10. Copy the Key 1 value into Notepad.

Note: We'll need this key later on for text translation.

12.3 Add Translation Support to the BOT

1. Open the Startup.cs file.
2. Find the comment // Add the translator speech middleware and add after the following code snippet:

```
3. services.AddSingleton(new TranslatorSpeechMiddleware(
4.     Configuration["SpeechSubscriptionKey"],
5.     Configuration["TranslatorTextSubscriptionKey"],
6.     Configuration["Region"],
```

```
conversationState.CreateProperty<ReservationData>(EchoBotAccessors.ReservationStateName)));
```

8. Open AdapterWithErrorHandler.cs.
9. Replace the constructor header to be:

```
public AdapterWithErrorHandler(ICredentialProvider credentialProvider, ILogger<BotFrameworkHttpAdapter> logger, PersonalityChatMiddleware personalityChatMiddleware, TranslatorSpeechMiddleware translatorSpeechMiddleware, ConversationState conversationState = null)
    : base(credentialProvider)
```

12. Find the comment `// Use translator speech middleware` and add the following code snippet:

```
if (translatorSpeechMiddleware == null)
{
    throw new NullReferenceException(nameof(translatorSpeechMiddleware));
}

Use(translatorSpeechMiddleware);
```

19. Open the appsettings.json file.
20. Replace the <your_speech_subscription_key> and <your_translator_text_subscription_key> values with the keys previously obtained from Azure.
21. Open the EchoBot.cs file.
22. Modify the `OnTurnAsync` method, and in the switch statement `switch (topIntent)` add the following code snippet before the default case:

```
case "GetDiscounts":
    await GetDiscountsHandlerAsync(turnContext);
    break;
```

26. Add the following method to the class:

```
27. private async Task GetDiscountsHandlerAsync(ITurnContext context)
28. {
29.     var msg = "This week we have a 25% discount in all of our wine selection";
30.     await context.SendActivityAsync(msg);
31. }
```

12.4 Test the Speech Translation

Deploy and test translator speech:

1. Go to Visual Studio and Right-click the ChatBot project.
2. Click Publish.
3. Click Publish again and wait for the deployment to complete. This step might take a few minutes.
4. Open your chatbot in a browser: https://chatbot-<your initials>.azurewebsites.net/.
5. Click the Speech option .
6. Check the Test bot translator with an audio file option.
7. Click the Browse button and select the file Downloads/buildanintelligentbot/resources/discounts_french.wav.
8. [] Select the French language from the drop down list.

   Note: If you want to see all the languages available, you can use the languages endpoint: https://dev.microsofttranslator.com/Languages?api-version=1.0&scope=text,speech,tts
9. Click Submit Audio and wait for the BOT to respond back.
10. The BOT will respond with audio and the text "Cette semaine, nous avons un rabais de 25% dans l'ensemble de notre sélection de vins".

Chapter 13: Adding a BOT to Web Application

Through this chapter we aim to make the students:

- Understand the concept of Web Chat libraries.
- Incorporate the different components in to the BOT.

13.1 Add the Web Chat Libraries

1. Find Downloads\buildanintelligentbot\resources\restaurant-lab.zip and uncompress it in the same directory.
2. Add the following snippet before the custom web chat stylesheet bot.css:
3. <link href="./lib/botchat.css" rel="stylesheet" />
4. Add the following code snippet right before the closing HEAD tag: </head>.
5. <script src="./lib/botchat.js"></script>
   <script src="./lib/CognitiveServices.js"></script>

13.2 Add the Web Chat Widget

Add the Bot Framework Web Chat control to the static page index.html.

```
<div id="bot"></div>
```

13.3 Add the Initialisation Script

Add the following code snippet immediately after the Web Chat DOM element added in the previous step:

```
<script>
 const speechSubscriptionKey = <your-speech-subscription-key>;

 const user = {
  id: 'userid',
  name: 'username',
 };
```

```
const bot = {
  id: 'botid',
  name: 'botname',
};

const speechRecognizer = new CognitiveServices.SpeechRecognizer({
  subscriptionKey: speechSubscriptionKey,
});

const speechSynthesizer = new CognitiveServices.SpeechSynthesizer({
  subscriptionKey: speechSubscriptionKey,
  customVoiceEndpointUrl: 'https://eastus.tts.speech.microsoft.com/cognitiveservices/v1',
});

const speechOptions = {
  speechRecognizer,
  speechSynthesizer,
};

BotChat.App({
    bot: bot,
    locale: 'en-us',
    user: user,
    speechOptions: speechOptions,
    directLine: {
      secret: <direct-line-secret>,
      webSocket: true,
    },
  },
  document.getElementById('bot')
);
```

```
  </script>
```

> **Note:** Replace `<your-speech-subscription-key>` and `<direct-line-secret>` with the same values previously obtained.

13.4 Test the BOT Using the Web Chat Widget

1. Open index.html in a browser.
2. Click on the microphone icon and talk to your BOT: "What is today's specialty?".
3. The BOT should respond back, now you are able to interact with your existing BOT from a new web site.

Chapter 14: Consuming a Custom Voice

Through this chapter we aim to make the students:

- Understand the concept of Custom Voice and voice fonts.
- Understand the process of incorporating voice fonts into BOTs, completing the integration process.

Custom Voice provides a way to create custom voices used to generate audio (text-to-speech). The custom voices are deployed as fonts, the models are called Voice Fonts. For this lab, we are using an existing custom voice that is provided by the Speech service. We will also show you how to create your custom voice fonts and use them in your BOT.

14.1 Create a Voice Font

In order to create a voice font, you'll need to make a studio recording and upload the associated scripts as the training data. The service then creates a unique voice model tuned to your recordings. You can then use this voice font to synthesisze speech.

1. Prepare your files: A voice training dataset consists of a set of audio files, along with a text file containing the transcripts of all of these audio files. Each audio file should contain a single utterance.
2. After preparing the audio files archive and transcripts, upload them via the Custom Voice Service Portal available at https://cris.ai/Home/CustomVoice. This portal will allow you to create the voice dataset and model.
3. Follow the instructions available at https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-customize-voice-font to create your new voice and deploy your model to an endpoint.
4. Once you create your custom endpoint, copy the Endpoint Url and the Font Name as you will need them later.

14.2 Add the Voice Font to your BOT

Let's modify our BOT code to use a new custom voice. If you don't have a custom font, you can use one from the Text-to-Speech API (https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/supported-languages#text-to-speech).

Add the Voice configurations:

1. Open appsettings.json and replace:
   - The *VoiceFontName* value with the one previously obtained.
   - The *VoiceFontLanguage* value with the language of your recording.

Note: Click on the following link https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/supported-languages to see more information about supported messages in Speech Service.

1. Open default.htm, go to the line 111 and replace the *customVoiceEndpointUrl* with your endpoint URL.

Deploy and test the voice font:

1. Go to Visual Studio and Right-click the ChatBot project.
2. Click Publish.
3. Click Publish again and wait for the deployment to complete. This step might take a few minutes.
4. Open your chatbot in a browser: https://chatbot-<your initials>.azurewebsites.net/.
5. Click the Speech option and say: I need a table reservation.
6. The BOT should respond using the new voice.

Module 9: Create LUIS App without Coding

Chapter 15: Web-based LUIS

Through this chapter we aim to make the students:

- Understand LUIS and integrate the components to the BOT.

15.1 Introduction to LUIS

Language Understanding (LUIS) is an API service based on the cloud system applying custom machine learning intelligence to a user's conversational, natural language text. This is to predict the overall meaning in the given context and pull out relevant, detailed information. A client application for LUIS is any conversational application that communicates with a user in natural language to complete a task. Examples of client applications include social media apps, chatbots and speech-enabled desktop applications.

Language Understanding (LUIS) as an API has many features that make it quite versatile and user-friendly. Here are a few important features:

Features

- Add Natural Language to your Apps – It is designed to interpret the user goals (intents) and distill the required information from the sentences (entities), to result in a high quality, nuanced language model. It also integrates seamlessly with the Azure Bot Service, making it easy to create a sophisticated BOT.
- Quickly Build a Custom Language Solution – Pit combines powerful developer tools with customisable pre-built apps and entity dictionaries, such as Calendar, Music, and Devices for building and deployment of solutions. The collective knowledge of the web and supply billions of entries is mined to form Dictionaries. These helps the model to correctly identify valuable information from user conversations.
- Learning and Improving – It can continuously improve the quality of the natural language models via active learning.
- Scalable – It is ready to be deployed in commercial applications and can scale with enterprise quality and performance. The service meets international compliance standards and supports 13 languages, making it highly accessible around the world.

15.2 Define your App

A LUIS app schema contains intents and entities relevant to your subject domain. The intents classify user utterances and the entities extract data from the user utterances. LUIS offers pre-built domains for many common scenarios. Check to see if you can use a pre-built domain as a starting point for your app.

Domain Identification

A LUIS app is centred around a subject domain. For example, you may have a travel app that handles booking of tickets, flights, hotels and rental cars. Another app may provide content related to exercising, tracking fitness efforts and setting goals. Identifying the domain helps you find words or phrases that are relevant to your domain.

Intent Identification

An intent represents a task or action the user wants to perform. It is a purpose or goal expressed in a user's utterance.

Utterances are inputs from the user that your app needs to interpret. To train LUIS to extract intents and entities from them, it's important to capture a variety of different example utterances for each intent. Active learning, or the process of continuing to train on new utterances, is essential to machine learning intelligence that LUIS provides. It is necessary to think of utterances that users might enter. Include utterances, which mean the same thing, but are constructed in a variety of different ways:

- Utterance length - short, medium, and long for your client-application.
- Word and phrase length.
- Word placement - entity at beginning, middle, and end of utterance.
- Grammar
- Pluralisation
- Stemming
- Noun and verb choice
- Punctuation - a good variety using correct, incorrect and no grammar.

Think about the intents that are important to your application's task.
Let's take the example of a travel app, with functions to book a flight and check the weather at the user's destination. You can define the `BookFlight` and `GetWeather` intents for these actions. In a more complex app with more functions, you have more intents and you should define them carefully so the intents aren't too specific. For example, `BookFlight` and `BookHotel` may need to be separate intents, but `BookInternationalFlight` and `BookDomesticFlight` may be too similar.

When choosing utterances, be aware that what you think is a common term or phrase might not be correct for the typical user of your client application. They may not have domain experience. Be careful when using terms or phrases that a user would only say if they were an expert. When planning your app, consider whether or not you use Bing Spell Check to correct user input before passing it to LUIS. If you do not spell check user utterances, you should train LUIS on utterances that include typos and misspellings. Here are a few examples of utterances:

Travel app intents	Example utterances
BookFlight	"Book me a flight to Rio next week" "Fly me to Rio on the 24th" "I need a plane ticket next Sunday to Rio de Janeiro"
Greeting	"Hi" "Hello" "Good morning"
CheckWeather	"What's the weather like in Boston?" "Show me the forecast for this weekend"

LUIS contains pre-built domains for the users to ease build-up and fasten the process. To select pre-built domain, you have to open your app by selecting its name on My Apps page. Select Prebuilt Domains from the left toolbar. Find the domain you want added to the app then select Add domain button.

Intent Scoring

You assign an utterance to a single intent. When LUIS receives an utterance on the endpoint, by default, it returns the top intent for that utterance. If you want the scores for all intents for the utterance, you can provide a flag on the query string of the prediction API.

Prediction API version	Flag
V2	`verbose=true`
V3	`show-all-intents=true`

The intent represents action the application should take for the user and is based on the entire utterance. An utterance can have only one top scoring intent, but it can have many entities.

Create an intent when the user's intention would trigger an action in your client application, like a call to the `checkweather()` function. Then create entities to represent parameters required to execute the action.

Intent	Entity	Example utterance
CheckWeather	["type": "location", "entity": "seattle"] ["type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution":"2018-05-23"]	What's the weather like in `seattle tomorrow`?
CheckWeather	["type": "date_range", "entity": "this weekend"]	Show me the forecast for `this weekend`

LUIS stores all logged utterances in the query log, available for download on the LUIS portal from the Apps list page, as well as the LUIS authoring APIs at https://westus.dev.cognitive.microsoft.com/docs/services/5890b47c39e2bb17b84a55ff/operations/5890b47c39e2bb052c5b9c2f

15.3 Train and Test the Model

Training is the process of teaching your Language Understanding (LUIS) app to improve its natural language understanding. Train your LUIS app after updates to the model such as adding, editing, labeling or deleting entities, intents or utterances. Training and testing an app is an iterative process. After you train your LUIS app, you test it with sample utterances to see if the intents and entities are recognised correctly. If they're not, make updates to the LUIS

app, train and test again. Training is applied to the active version in the LUIS portal. To start the iterative process in the LUIS portal, you first need to train your LUIS app at least once. Make sure every intent has at least one utterance before training. Access your app by selecting its name on the My Apps page and then in your app, select Train in the top panel.

When training is complete, a notification appears at the top of the browser. Training date and time are GMT + 2. Training uses a small percentage of negative sampling. If you want to use all data instead of the small negative sampling, use the API. Use the Version settings API with the `UseAllTrainingData` set to 'true' to turn off this feature. You do not need to train after every single change. Training should be done after a group of changes are applied to the model and the next step you want to do is to test or publish. If you do not need to test or publish, training isn't necessary. Training in the LUIS portal is a single step of pressing the Train button. Training with the REST APIs is a two-step process. The first is to request training with. Then request the training status with HTTP Get. In order to know when training is complete, you have to poll the status until all models are successfully trained.

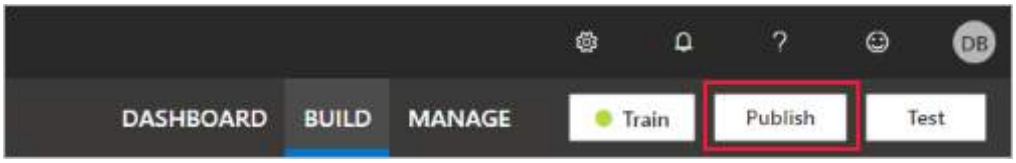
Testing is the process of providing sample utterances to LUIS and getting a response of LUIS-recognised intents and entities. You can test LUIS interactively, one utterance at a time, or provide a list of utterances. While testing, you can compare the current active model's prediction response to the published model's prediction response.

- Interactive Testing: It is done from the Test panel of the LUIS portal. You can enter an utterance to see how intents and entities are identified and scored. If LUIS isn't predicting the intents and entities as you expect on an utterance in the testing panel, copy it to the Intent page as a new utterance. Then label the parts of that utterance for entities and train LUIS.
- Endpoint Testing: You can test using the endpoint with a maximum of two versions of your app. With your main or live version of your app set as the production endpoint, add a second version to the staging endpoint. This approach gives you three versions of an utterance: the current model in the Test pane of the LUIS website and the two versions at the two different endpoints.

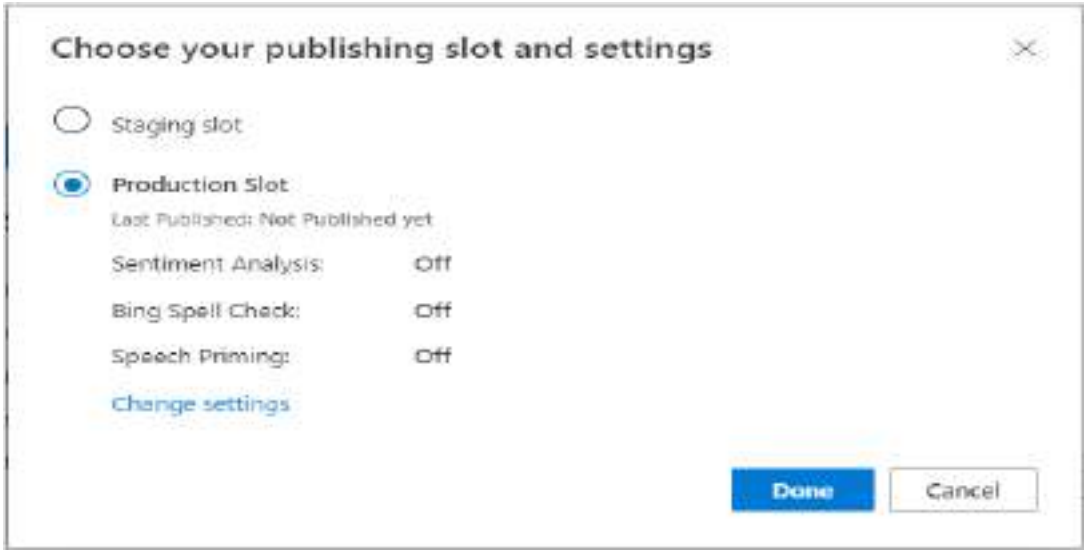
15.4 Publish the App and Generate JSON Code

In order to receive a LUIS prediction in a chatbot or other client application, you need to publish the app to the prediction endpoint. Here are the steps:

Step 1: Select Publish in the top-right navigation.



Step 2: Select the Production slot, then select Done.



Step 3: Select the Access your endpoint URLs link in the notification to go to the Azure Resources page. The endpoint URLs are listed as the Example Query.

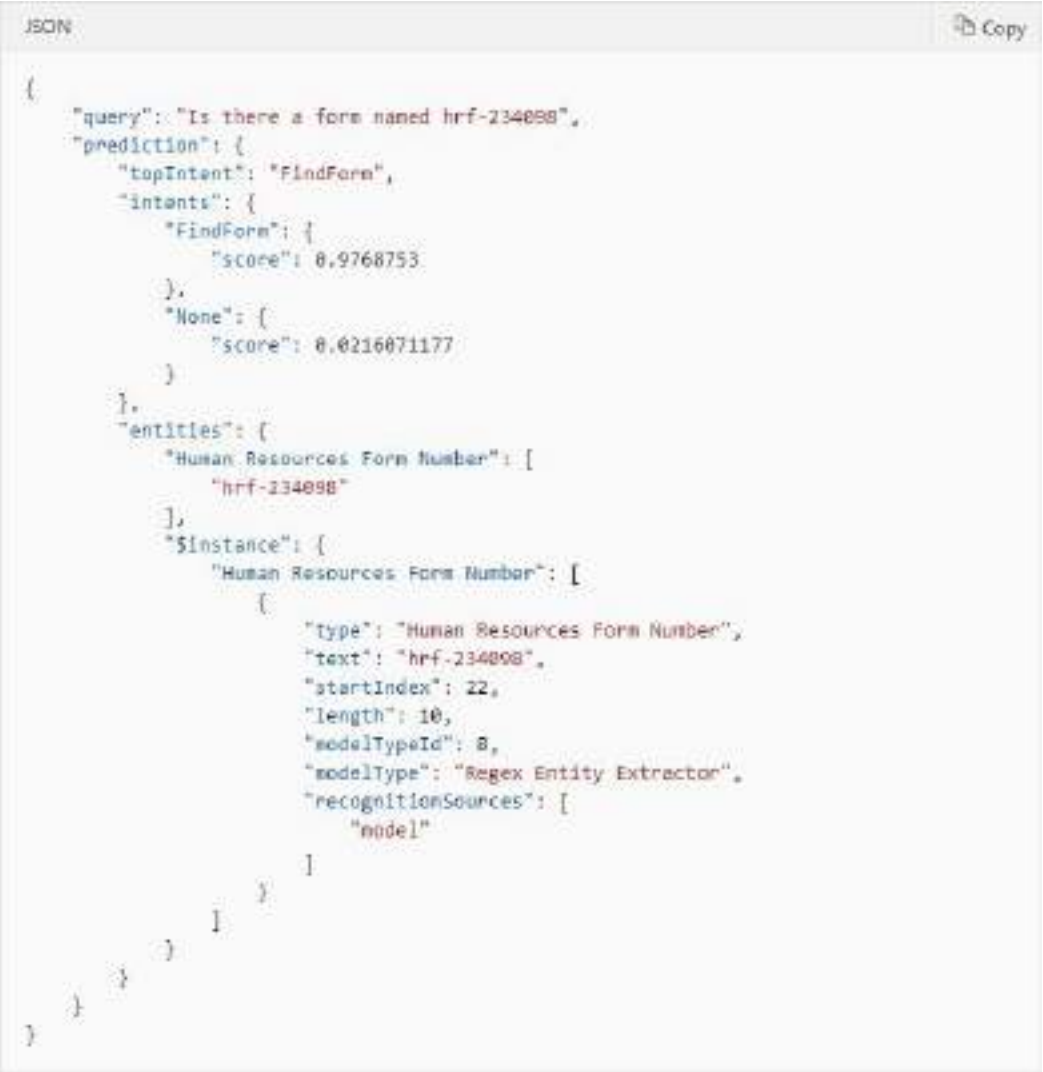
Step 4: In the portal, `query=` at the end of the URL is where the user's utterance is appended to the GET request. After the `query=`, enter the same user utterance used at the end of the previous quickstart:

Is there a form named `hrf-234098`

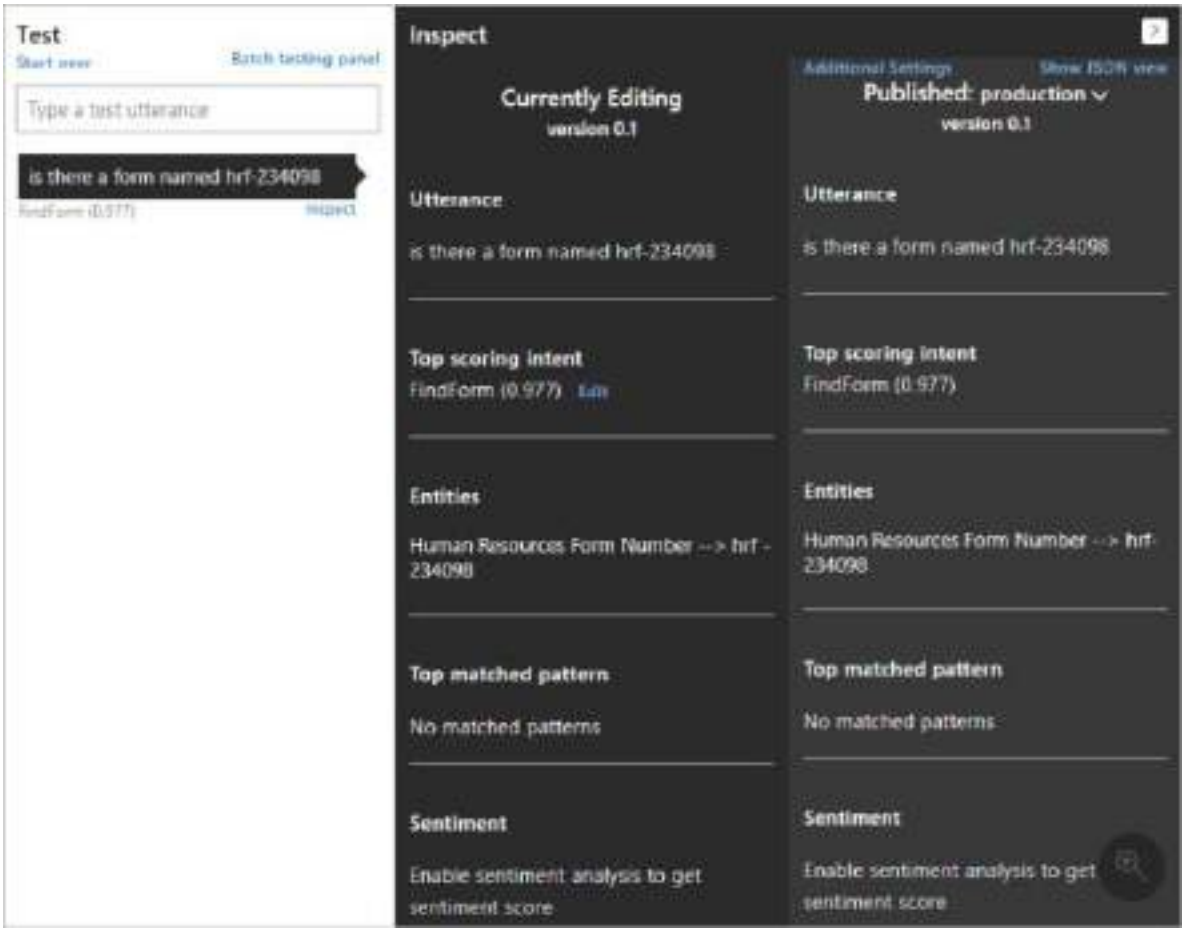
Make sure the query string includes the following pairs:

- `show-all-intents=true`
- `verbose=true`

The browser shows the following response:



To see this same level of information in the test pane, you must publish the app. After the app is published, select Compare with published in the test pane. Use Show JSON view in the published test pane to see the same JSON as the previous step. In this way, you can compare changes to the current app you're working on with an app that is published to the endpoint.



15.5 Assess Your Knowledge

15.5.1 Practical Assignment and Lab Work

Create intents and utterances on the following actions:

- Booking a cab from your home to the nearest market.
- Checking the prices of a tee-shirt in your favourite colour.
- Searching for a book by your favourite author.

15.6 Reference Links

- https://luis.ai

Module 10: Project Work

Practical Assignments/ Project Work

Assignment 1: Create a bank of images (three sets of 15 images each, consisting of pictures with various emotions). Using Azure Suite of applications select the best suitable application to read the images and analyse the same.

Assignment 2: Write down a speech of your own or download any speech of a public personality of your choice. Using Azure Suite of applications, select the best suitable application to correct the speech in terms of the language, choice of words and grammar. Use the most suitable application to convert the same into three different languages of your choice.

Assignment 3: Design the user interface for a project of your choice (library management system, attendance management system etc.). You can make multiple user interfaces of the project. Using Sketch to Code create the front end screens (user interface) of the project.

Assignment 4: Create a bank of images (more than 50). It should contain images of people with emotions (various ages, colour, expressions etc.). Using the 'Face and Emotion Recognition' application in conjunction with the image processing application, run the images to predict the emotion and analyse visual content.

Assignment 5: Imagine a hypothetical situation, where you are looking forward for various applicable career choices after standard 10^{th} with the help of a chatbot assisting you in the process. Prepare question answer trail using relevant Azure services.

Assignment 6: Create utterances on the following actions:

- Checking the weather in your city.
- Searching the cricket score of your favourite player.

Assignment 7: Create intents on the following actions:

- Checking the weather in your city.
- Searching the cricket score of your favourite player.

Assignment 8: Create entities on the following actions:

- Checking the weather in your city.
- Searching the cricket score of your favourite player.

Assignment 9: With the help of an example demonstrate the addition of utterances in LUIS.

Assignment 10: Create an example to show integration of QnA with LUIS.

In addition to the above mentioned, here are a few real-time assignments mentioned in the table below:

Scenario	Overview	Features Covered	Need Webcam
Anomaly Detector	Showcases anomaly detection with three pre-selected data streams, as well as a Live Sound scenario that samples the audio level of your microphone as the input data stream.	Anomaly Detector API	No
Insurance Claim Automation	An example of Robotic Process Automation (RPA), leveraging Custom Vision and Form Recognizer to illustrate automating the validation of insurance claims.	Custom Vision API, Form Recognizer API, Bing Image Search API, Bing AutoSuggestion API	No
Real-time Image Classification	A real-time workflow for processing frames from a web camera to show classification models exported by Microsoft Custom Vision Service.	Custom Vision API, Windows Machine Learning	Yes
Real-time Object Detection	A real-time workflow for processing frames from a web camera to show object detection models exported by Microsoft Custom Vision Service.	Custom Vision API, Windows Machine Learning	Yes
Real-time Video Insights	A real-time workflow for processing frames from a video file to derive real-time insights such as demographics, emotion, unique face counting and face identification.	Real-time sampling, Age, gender and emotion prediction, Face identification, Unique face counting, Celebrity detection	No

Glossary

Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

API: A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application or other service.

Bing: A kind of search engine; a heap.

Code: Program instructions.

Internet: A global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardised communication protocols.

Objective: A thing aimed at or sought; a goal.

Predictions: The action of predicting something.